Organization of No. 1 ESS Central Processor

By J. A. HARR, F. F. TAYLOR and W. ULRICH

(Manuscript received January 28, 1964)

The central processor controls the operation of the No. 1 electronic switching system by executing sequences of program instructions. The logical organization of the central processor is described by the simultaneous evolution of:

(1) an instruction repertoire to carry out the required telephone and system maintenance tasks efficiently, and

(2) a circuit logic design to provide the necessary circuits (flip-flop registers, accumulators, etc.) to execute the instructions at a high data processing rate.

The design aims, order structure, timing, internal sequencing, and communications with the peripheral equipment of the system are described.

1. INTRODUCTION

Telephone central offices must cover a wide range of sizes and provide a large variety of services to customers; in addition, they must be compatible with existing systems, adaptable to varied and changing operating conditions, dependable, reliable, and economical. The development of an electronic switching system capable of satisfying these requirements presented many new problems to the designers. As a result, many techniques new to the telephone switching field were introduced in the system.[1] One of the most important new techniques is the control philosophy, which utilizes a stored program.

A system employing a stored program is one which consists of memories for storing both instructions and data, and a logic unit which monitors and controls peripheral equipment by performing a set of operations dictated by a sequence of program instructions. The stored program philosophy permitted the designers to use centralized logic circuitry and large-capacity memory units as a means of attaining flexibility and over-all economy in the system.

In this paper some of the design characteristics of the central processor are described, followed by a step-by-step development of the central control order structure and the corresponding logic circuitry needed for its implementation.

II. DESIGN CHARACTERISTICS

A simplified diagram of the electronic switching system (ESS) is shown in Fig. 1. The outer circle represents the entire No. 1 ESS, having primary inputs from lines and trunks[2] via scanners,[3] and outputs to the network[4] and signal distributor,[3] with teletypewriters as administrative input-output devices and with a magnetic tape for automatic message accounting (AMA)[5] output. The inner circle, the central processor, contains a central control[6] unit which executes program instructions and memory units used for storing program instructions and data.

In order for the central processor to handle traffic submitted by offices

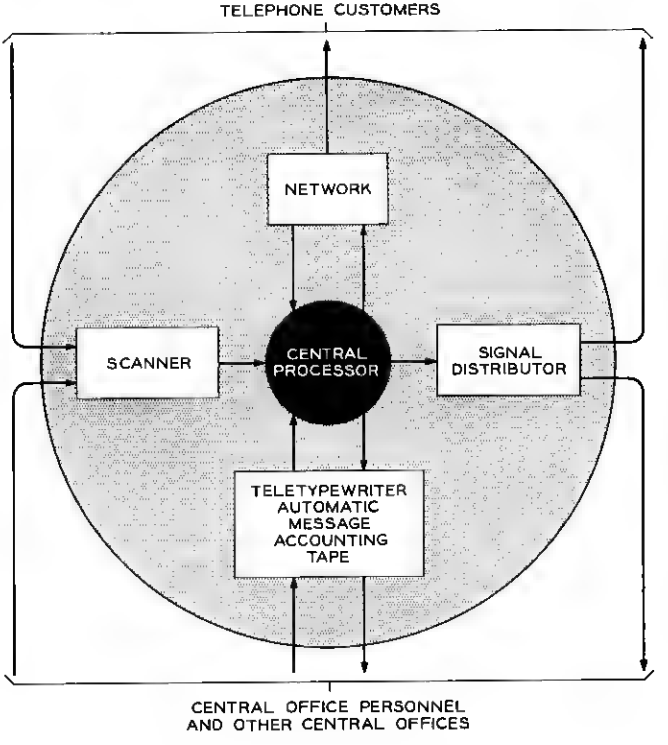


Fig. 1 — Simplified diagram of the No. 1 ESS.

serving 5000 to 60,000 customers, the number of memory units in the system must be expandable over a wide range. Therefore the central processor address registers, memory word size, and address buses are designed to accommodate the largest office. The system design must not only meet initial office size requirements, but must also include growth capability.

Since the data processing operations required for performing telephone functions are accomplished by executing a stored program[7,8] which must be error-free and remain error-free* at all times to insure the proper behavior of the system and good service for the customers, the memory chosen for storing the program is semipermanent and requires off-line operations to change it. (This avoids the risk of an error in operation introducing an error in the program.) Therefore the central processor contains two types of memory: a semipermanent memory system (program store)[9] for storing programs and a high-speed readable and writable memory (call store)[10] for storing call progress data.

An address of at least 21 binary bits is needed to gain access to the total number of words required in both the program and temporary memories. To meet this requirement, along with memory store design considerations, the designers decided upon a word length in the call store of 24 bits, comprising a parity check bit and 23 information bits. For convenience, the same word length is used throughout the central control. Most instructions which operate on memory words require a 21-bit address field. Instructions which contain a data field must allow for 23 bits in the data field to be compatible with the length of temporary memory words. Accordingly, the operation fields of each of these types of instructions are 16 and 14 bits in length, respectively, to accommodate the many types of instructions needed in each category. Therefore the instruction word length is 37 bits. To check and correct program instructions with single errors and to determine whether a word read has a double error in it, 7 check bits are also needed for each program instruction, making a total of 44 bits stored for each entry in the program store.

Engineering studies of the number and kind of telephone functions which the system must perform dictated a need for an efficient instruction repertoire with specific attributes. Some of these attributes will now be described.

To perform the functions required to handle the busy-hour traffic submitted by the largest office, the system can spend only approximately

* As will be seen later, facilities are available for detecting and correcting single errors.

5000 machine cycles per call. To meet this requirement, the instruction repertoire must include efficient multifunctional program instructions.

Since the repertoire must include the storage of data of variable bit length in the temporary memory and retrieval of these data, both read-from-memory and write-to-memory instructions include masking facilities. As used here, the masking of words read from memory means changing all of the bits of the word to zero except those which specify the item of interest; these remain in the same state as they were in memory. Masking of words written into memory is a facility for inserting an item of variable length into a word already in memory; the remaining bits of the word are unchanged.

To assist in achieving program word efficiency, in most cases the same functional program is used for all calls in progress requiring the execution of a given function.[7,8] For reliability, instructions in this system are not changeable at run-time (i.e., while the machine is actually processing calls). Indexing facilities provide means by which the programmer can change the addresses of memory words acted upon by the same program. Also, the indexing facilities can be used to vary the sequence of programs to be read and executed.

To carry out data processing of call information, the repertoire includes the arithmetic operations of addition, subtraction, comparison, shifting, and rotation, and the following logic operations: AND, OR, EXCLUSIVE-OR, and COMPLEMENT. Since the functions which the system must perform do not require multiplication and division operations, it was not necessary to include these in the central processor. To perform the logic functions required to carry out the many telephone functions, a variety of decision instructions are required to transfer control to specific program sequences based on the condition of internal central control registers after data manipulations have been performed on them. For example, a program of three instructions capable of performing the following three operations illustrates the primary way the central processor can vary the sequence of its operations according to input data it has received.

(1) Read, at an address specified by an index register, the word from memory containing the first dialed digit of a call; mask out all of the bits in the word other than the four bit positions used for the first digit; and load the word into an accumulator register.

(2) Compare the word just loaded into the accumulator with the value 10 (i.e., the number of pulses counted when a customer dials zero) specified by the data field of the instruction.

(3) Transfer to the program specified in the address field of the in-

struction if the two compared quantities are equal (this program will cause the customer to be connected to an operator); otherwise, continue the present program sequence.

For efficient operation, the central control should be capable of executing instructions which combine a number of the operations listed above. For example, the repertoire includes an instruction which reads the word at a temporary memory address specified by an index register, masks the word read, complements it, and then AND's the result with the accumulator register in the central control during one operational cycle.

To make efficient use of data processing time the repertoire must include a class of special instructions designed to facilitate the reading and making of logical decisions on input data and special orders for delivering output data to both the network controllers and trunk control circuitry. Therefore the central control can be described as an input-output processor superimposed on a general-purpose data processor.

Since the fundamental task of the central processor is continuous monitoring and controlling of its rapidly changing environment, consisting of wide variations of traffic submitted by customer lines and trunks, real time must be carefully considered when planning the system and writing the program. Programs to monitor and gather input data and to deliver output signals and data must be especially efficient in their use of central processor cycle time.

Input-output programs must be executed on a strict schedule. For example, the program to detect and receive dial pulses must be performed every 10 milliseconds. In order to accomplish this, an interrupt mechanism is included in the central processor. The interrupt mechanism, when activated by a source such as a clock or check circuitry, generates a new program address, thus transferring control of the system to an interrupt program sequence. When this occurs, the address of the next instruction which normally would have been executed is automatically stored. When the interrupt program completes its task, control is returned to the interrupted program.

To make the system capable of continuous operation during malfunction of circuit components, the central processor is duplicated. In order to detect and ultimately pinpoint hardware malfunctions, the central processors include:

(1) circuitry which compares the execution of instructions in both central processors,

(2) circuitry for generating a parity bit for each word stored in the temporary memory,

(3) circuitry for checking the parity of words read from temporary memory,

(4) circuitry for detecting and correcting single-bit errors in instructions read from the program memory,

(5) circuitry within the peripheral equipment for checking data received and for notifying the central processor when either data it receives or its own check circuitry indicates trouble,

(6) circuitry within the central control for verifying signals sent back by peripheral equipment, and

(7) circuitry for special-purpose instructions for controlling and interrogating stores and peripheral equipment.

III. BASIC DESCRIPTION OF CENTRAL CONTROL

In this section an order structure, including the symbolic names of the instructions, and a central control block diagram will be concurrently explained.

3.1 *Basic Facilities*

In the simplest form (see Fig. 2), the central processor consists of a central control, a program store which receives an address from central control and returns the corresponding instruction, and a call store, which receives an address and either receives data to be recorded at that location or returns the data previously stored at that location.

As a starting point, central control must contain registers for receiving program store instructions and call store data, and facilities for generating and transmitting addresses to both stores and data to the call store (see Fig. 3). The buffer order word register (BOWR) receives instructions from the program store, and the data buffer register (BR) receives data from the call store. The program store address register (PAR) is

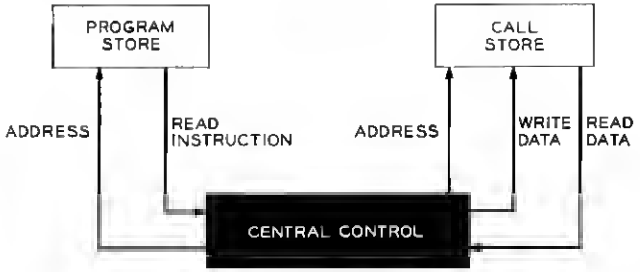


Fig. 2 — Block diagram of the data processor.

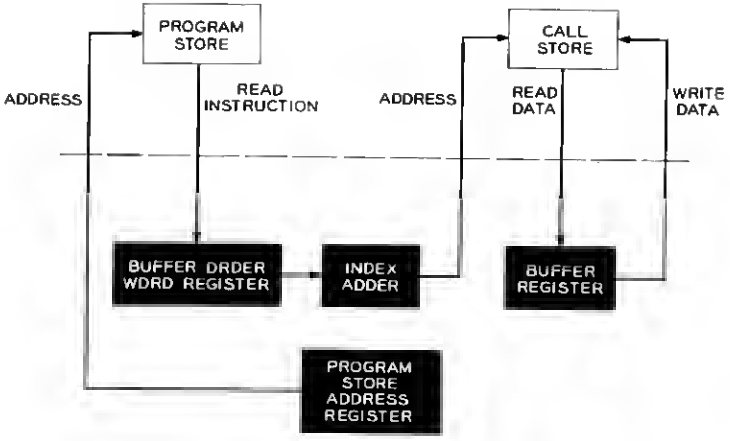


Fig. 3 — Derivation of the central control (1).

used as the source of addresses of instructions to the program store, and the index adder is used to generate addresses for the call store.

These facilities must be augmented by an instruction decoder, the buffer order word decoder (BOWD) attached to the buffer order word register (see Fig. 4). This decoder is used to control the gating of information inside central control.

Also needed to control such gating and to synchronize the decoding with the reading of information from the stores is a clock. In the No. 1 ESS central control, the clock is a synchronous 5.5-microsecond clock, with 22 distinct phases separated by 0.25 microsecond. Arbitrary-length clocking pulses are derived by setting a flip-flop circuit with one arbitrary phase, resetting it with another, and using the output of the flip-flop as the clocking pulse. Such pulses are repeated once every 5.5 microseconds.

Gates are therefore controlled by decoding the output of the buffer order word register and combining the decoded output with a suitable clock pulse.

3.2 *Index Registers*

In addition to the buffer register, there are a number of general-purpose flip-flop index registers, F, X, Y, and Z (see Fig. 4). The index registers are 23 bits long, the basic word length of the central control and the call store. (The 24th bit of the call store, a parity check bit, does not store useful information; this bit need not be carried along in central control data processing, although it must be generated anew when-

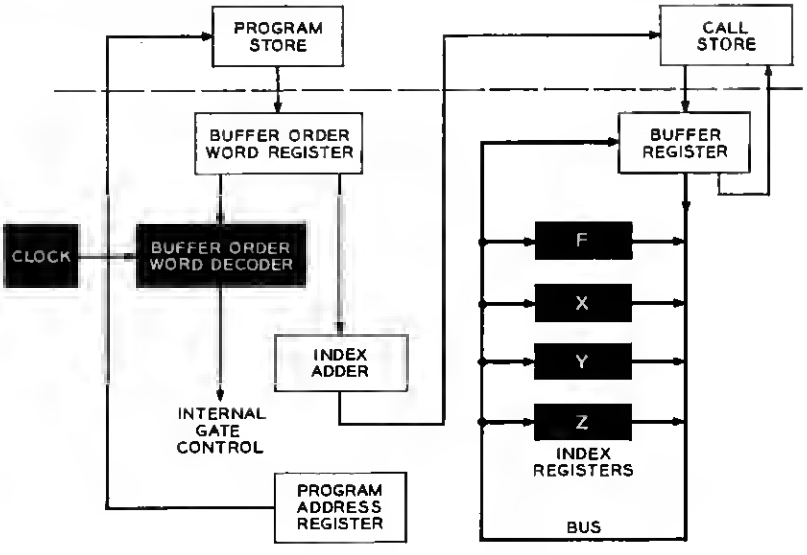


Fig. 4 — Derivation of the central control (2).

ever information is stored in the call store and checked whenever information is read from the call store.)

Indexing is useful for developing a program which is general for any telephone central office. It is the process of deriving a memory address by adding a constant from the instruction to a variable previously derived and stored in an index register. For example, the index register might contain the starting address of a block of call store words used for accumulating dialed information; the constant might be the location of a word within such a block, containing information known to be needed at a certain stage of a call. Such an instruction may be described as follows: fetch the call store reading in the third word of a block whose starting address is stored in the Y index register, and store the reading in the X index register. The description of the operation is divided into three parts (see Fig. 5): the basic operation (fetch data from memory

	OPERATION	CONSTANT	INDEX REGISTER	———
MEMORY TO X REGISTER	MX	3	Y	

Fig. 5 — Basic instruction format 1.

and store in the X register), the constant of the instruction (3), and the index register used in indexing (Y). In the basic mnemonics of the system, memory to X register is written as MX; therefore this instruction is written as MX,3,Y. Instructions which read or write in memory contain M in their mnemonic representation and are collectively designated M instructions.

3.3 *Bus*

Information is transmitted among the index registers via a bus (Fig. 4) consisting of 23 parallel information paths. The F, X, Y, and Z registers plus the buffer register, which can also be treated as an index register, all have output gates to and input gates from the bus.

3.4 *Index Adder*

In order to perform indexing, an adder (see Fig. 6) is required. The index adder receives one input (the constant of the instruction) from the buffer order word register, and the other input (the variable, i.e., the contents of the specified index register) from the bus. The output

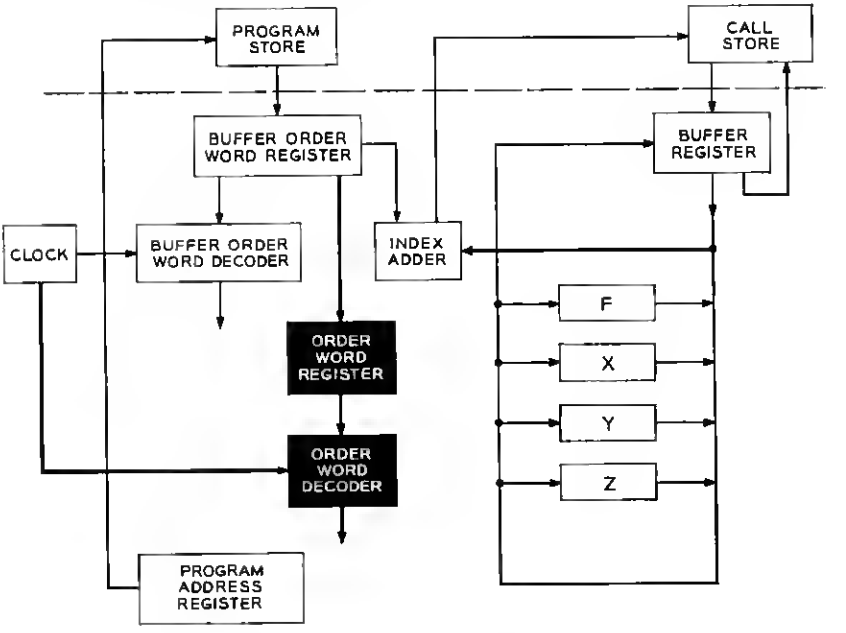


Fig. 6 — Derivation of the central control (3).

of the index adder, as previously indicated, is the source of addresses to the call store.

3.5 *Order Word Register*

The basic cycle time of the program store is 5.5 microseconds, and the maximum time from the reception of an instruction such as MX,3,Y until the specified reading from a call store is in the data buffer register is about 6.0 microseconds. When one considers the additional data processing of a call store reading after it has been received by central control, it can be seen that the execution time of an instruction occupies major fractions of two machine cycles. Thus there is overlap in the execution of two consecutive instructions. This overlap is described in Section VIII. An order word register and decoder are therefore provided to control part of the execution of an instruction. The buffer order word decoder and the order word decoder simultaneously control the execution of two consecutive instructions.

The buffer order word decoder controls the addressing of the call store. On a reading instruction, the order word decoder controls the gating of information from the call store to the data buffer register and thence, via the bus, to the destination register; on a writing instruction, the order word decoder controls the gating of data from some source register to the data buffer register and thence to the call store. The actions of the two decoders are sufficiently independent that the division of decoders into a buffer order word decoder and an order word decoder does not cost very much compared to the use of a single decoder.

3.6 *Masking: Logic Register, Unmasked Bus, Masked Bus, and Mask Circuit*

The 23-bit word length is much longer than many of the basic quantities of data. A long useful quantity of data is a 21-bit memory address. A typical short quantity is a single binary-coded decimal digit, 4 bits long; several such short quantities may be packed in a single word. In order to treat partial words efficiently, the central control has masking facilities (see Fig. 7). Since most data words pass over the bus, a single mask circuit on the bus accomplishes most of the masking functions. The mask circuit has two inputs, the unmasked bus, which is connected to the gates at the outputs of index registers, and the output of a logic register whose chief function is to control the masking function. The output of the mask circuit is called the "masked bus" and is connected to the input gates of index registers.

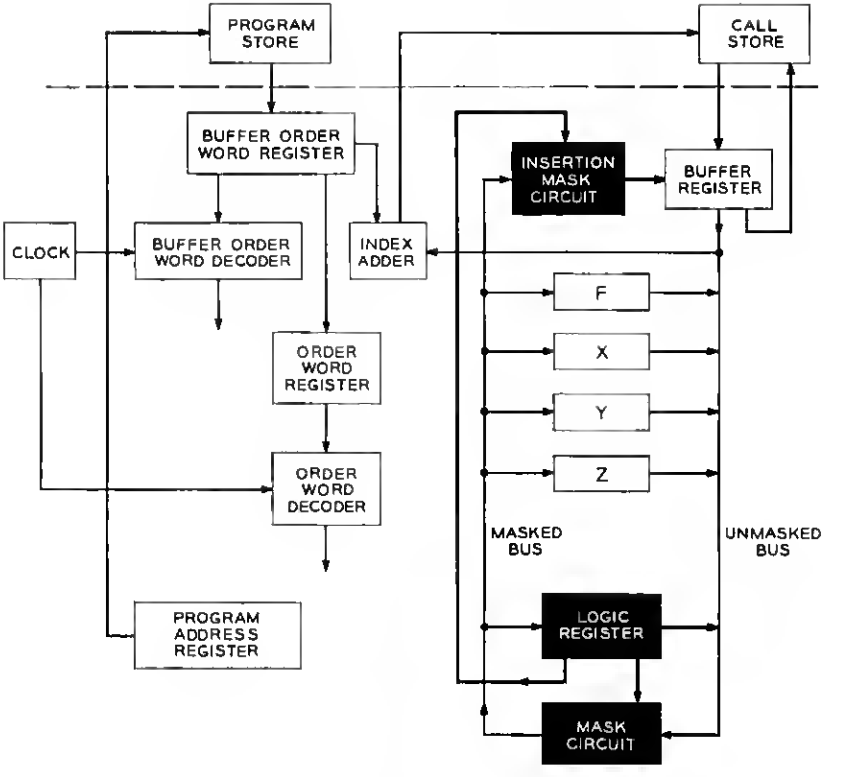


Fig. 7 — Derivation of the central control (4).

The logic register is 23 bits long; each bit controls the masking of 1 bit on the bus. The logic register is itself connected to the unmasked and masked bus so that it may be controlled and read as easily as any index register.

Masking is an option of most instructions. If in the previously described instruction (MX,3,Y) only the four least significant bits were of interest and the logic register were already set up with these four bits equal to one and the rest of the bits equal to zero, then by specifying masking only the four least significant bits would be transmitted to the X register (see Fig. 8). The rest of the bits would be transmitted as zero. This form of masking is called PL masking (P = product, L = state of the logic register; therefore, product with the state of the logic register). This instruction would then be written as MX,3,Y,PL.

	OPERATION	CONSTANT	INDEX REGISTER	MASK AND INSERTION
	MX	3	Y	PL (MASK)
X REGISTER TO MEMORY	XM	3	Y	EL (INSERTION)

PL OR EL WILL USE THE CURRENT CONTENTS OF THE LOGIC REGISTER L WHICH WAS SET BY A PREVIOUS INSTRUCTION

Fig. 8 — Basic instruction format 2.

3.7 *Insertion Masking*

Another form of masking that is used frequently is insertion masking. Insertion masking permits all but a selected group of the bits of a certain register to remain unchanged. The selected group of bits is then set up according to the instruction. Because of the requirement that certain bits remain intact, it is convenient to associate the insertion mask circuit with only one of the registers. The most logical choice is the buffer register, since insertion masking is most frequently used when only a portion of a word in the call store is to be altered. The insertion mask circuit is also controlled by the logic register, since in most cases the bits to be inserted and the position associated with these bits have been set up in the logic register for some previous masking (PL) operation. Insertion masking is indicated by specifying EL masking. (E = insertion, and L = the state of the logic register.) If, for example, the four least significant bits of the X register are to be stored in the address Y + 3* while leaving the other bits of that memory location intact, this action could be performed with the following two-step program (provided that the logic register is already set up to 1's in the four least significant bits and 0's elsewhere): MB,3,Y (read the contents of memory at the address Y + 3 into the data buffer register); XM,3,Y,EL (insert the contents of X into the BR for all bit positions of the logic register equal to one, leaving the rest of the bits of the BR intact; then write the buffer register into memory at address Y + 3). If PL, instead of EL, masking had been specified, the contents of memory would be all 0's except in the four least significant bits; by specifying EL the upper bits remain the same as they appear in the BR.

There is no circuitry available at the call store for performing the equivalent of insertion. Therefore, insertion into memory must always

* For simplicity, the following convention is used in this paper: the contents of a register, such as Y, plus a constant, such as 3, are represented by an unbracketed expression, such as Y + 3.

be a two-step operation: the first step to read the word at which information is to be inserted; the second step to insert this information and then write a complete word back. Insertion is entirely a central control function, not a store function.

3.8 *Transfer Facilities*

So far the details of addressing a program store have not been shown except for a program store address register (PAR) which transmits such an address. In the program, one of two things can happen. Normally, the program advances from one instruction to the next, so the contents of the program store address register are simply incremented by one. This is accomplished by attaching an increment circuit to the program store address register (see Fig. 9). (The program stores themselves do not have any incrementing facilities. A program store is always addressed with a complete address.) However, sometimes in a program a transfer is necessary. A transfer instruction is an instruction which causes the program to go to another set of instructions, not the immediately following instruction. The most convenient source of the address to which the program would transfer is the output of the index adder, since this is the place where the contents of the instruction are combinable with the contents of registers and thus indirectly with memory readings. A connection from the index adder to the program address register is therefore provided (see Fig. 9).

Direct transfers are transfers to an indexed address. Indirect transfers — i.e., transfers to an address stored in memory, the memory being read through the use of an indexed address — are also possible in No. 1 ESS. An indirect transfer is indicated by an M suffix in the index register field. An input (for simplicity, left out of the diagram) from the call store to the buffer order word register transmits the transfer address to the index adder, thence to the program store address register.

3.9 *Complement Option*

Another option existing in the system is the complement option. When numbers are considered numerical rather than logical data, a negative number is stored as the complement of the positive number whose absolute value is the same. The most significant bit, 22, is the sign bit of the entire quantity. This means that both a positive and a negative quantity 0 exist in the system, since the complement of all 0's (+0) is all 1's (−0). Such a system has the advantage of having very simple adder circuits, even though it does introduce occasional programming

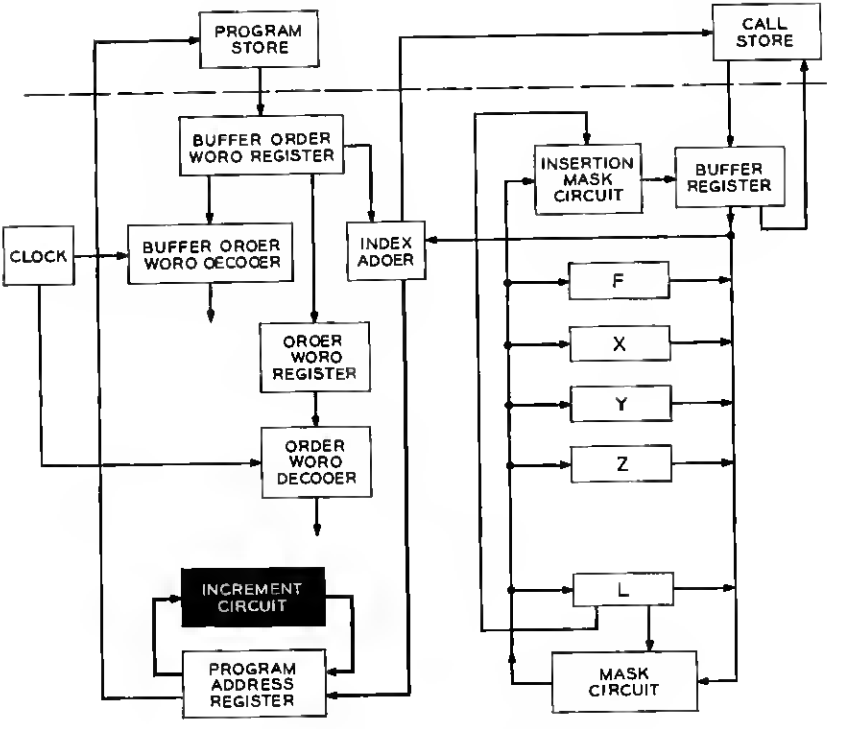


Fig. 9 — Derivation of the central control (5).

problems. The complement circuit is in series with the mask circuit (see Fig. 10); masking takes place before complementing. The complementing is specified as part of the mask field. Thus if we wish to load into X the masked and complemented contents of memory found at the address Y + 3 we may specify this instruction (see Fig. 11) by writing MX,3,Y,PLC.

3.10 *Data Instructions*

So far only instructions which deal with memory readings have been considered. There is another large class of instructions which deal with internal data manipulations and with the setting up or altering of registers by some constant (data word) within the instruction. These instructions are defined as W (for word) instructions. (It is important to bear in mind the fundamental property that instructions in this machine are not variable. A constant in an instruction is truly a constant until such a time as the program itself is altered, which can be done only by

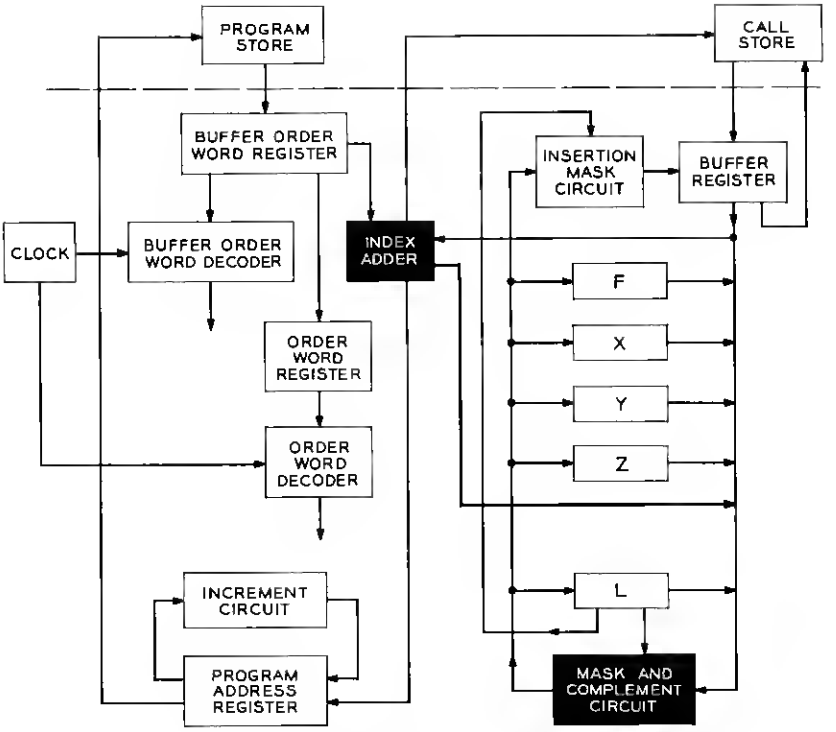


Fig. 10 — Derivation of the central control (6).

writing new permanent magnet twistor cards.) A natural channel for performing such data manipulations is via the index adder. Therefore the index adder has an output onto the unmasked bus (see Fig. 10). Thus, for example, we can increment register X by a constant by gating the X register to the index adder, adding the constant of the instruction and gating the output of the index adder via the bus back into the X register. All these operations are performed by the instruction WX (W equals indexed word, i.e., output of the index adder). This instruction is executed by circuit actions equivalent to generating a mem-

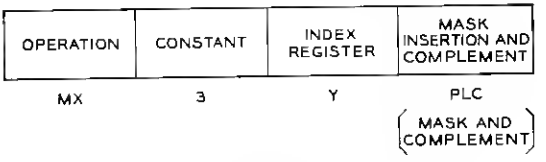


Fig. 11 — Basic instruction format 3.

ory address, except that the address is gated to the unmasked hus instead of the memory.

W instructions are also maskahle, since the output of the index adder has to go through the mask circuit hefore it arrives at the destination register. Thus the instruction WX,3,Y,PL takes the Y register, increments it by 3, and places the result in the X register after first masking it according to the present contents of the logic register.

3.11 *Accumulator*

The central control must perform many additions and logical comhinations of two quantities. It is convenient to use one register as an accumulator (K) and to permit this register to he comhined readily with masked memory and W-type data. The accumulator adder (see Fig. 12)

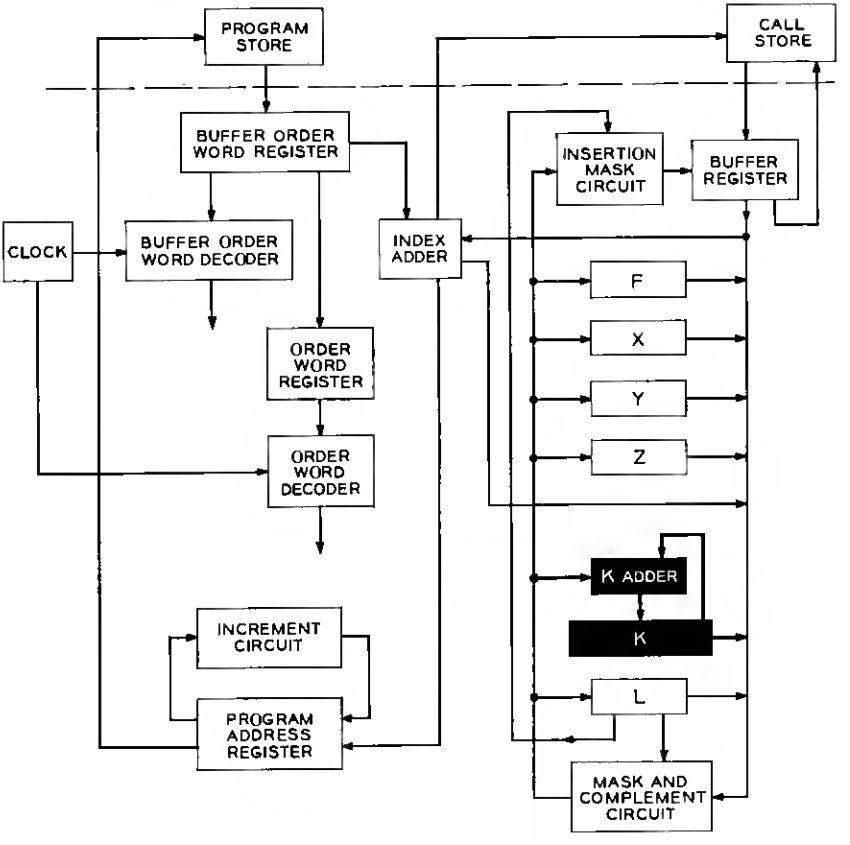


Fig. 12 — Derivation of the central control (7).

is capable of combining the present contents of the accumulator with indexed data or memory readings, both optionally masked, by adding, ANDing, ORing, or EXCLUSIVE-ORing the two. The timing problems are sufficiently severe that a single adder system cannot serve both as an index adder and an accumulator adder for combining two data operands. Accordingly, central control contains two adder systems to handle both operations concurrently.

Data in the accumulator can also be shifted and rotated. The shifting is not usually done for multiplication purposes, but to line up two items of information found in different positions within two data words to a position where the two items may be logically combined or treated in some other standard manner. For example, a pulse count for a decimal digit may always be accumulated in bit positions 0 through 3 (see Fig. 13). However, it may have to be stored in positions 4 through 7, or 8 through 11, or 12 through 15, according to which digit of a number it represents. To get data accumulated in positions 0 through 3 to positions 4 through 7, a shifting operation is necessary.

The rotation operation is similar to the shift operation except that for a left rotate the contents of the most significant bit, instead of being shifted out, are shifted back into the least significant bit, and vice versa for a right rotate. A special-purpose rotation within 16 positions of K is also available in central control. This rotation is extensively used in the network path hunt program.

Shifting is also performed very frequently when a number is composed of two parts, the first part indicating the location of an appropriate table of information and the second part indicating the location within that table.[11] For example, a line equipment number consists of a line link network and line switch frame indication, which is used to find a table, each table containing line translation information for one line switch frame and a position within that line switch frame which is used to find the line translation information within such a table. Without the ability to split such numbers into parts, it would be very diffi-

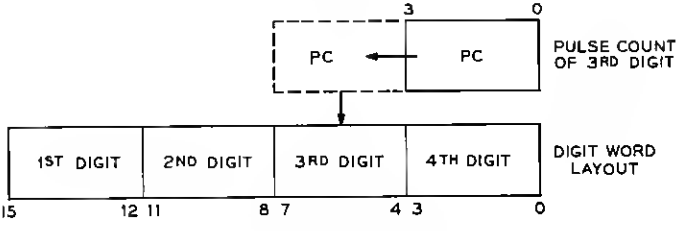


Fig. 13 — Use of shifting.

cult to organize the memory layout of the system for both rapid access and compact storage.

3.12 *Data from the Program Store*

So far the simplifying assumption has been made that data always come from a call store and instructions always come from a program store. In practice, however, much of the data is stored in the program store — specifically, the translation data. The process of reading a program store for data is a complicated one, especially in view of the overlap operation that is used.

The same types of instructions are used to read data from the program store and from the call store. This helps programming, since it does not fix a memory location at the time the program is written and helps to relieve the programmer from the burden of considering two different types of memories. A memory address decoder (see Fig. 14) connected to the output of the index adder recognizes when the output of this adder specifies an address that is not in the call store but is in program store.* It triggers a sequencer (see Fig. 14) which takes care of a special group of operations to be described below.

A sequencer is necessary to prevent the data that are coming from the program store from being incorrectly interpreted as an instruction. This sequencer must cause the program store to be read at the address specified by the output of the index adder and must then go on to the next instruction.

Fig. 15 shows the contents of the buffer order word register, order word register and program store address register during the processing of an instruction for reading data from the program store. The instruction is MX,BB,Y.† Y + BB specifies an address which happens to be in the program store. This instruction is located at the address AA. At time 1 the buffer order word register contains the instruction at address AA, the order word register contains the instruction at address AA − 1 and the program store address register contains the number AA + 1 in preparation for the reading of the next instruction. At time 2 the buffer order word register has the instruction at address AA + 1 but it cannot execute this instruction because the data word called for by the previous instruction has not yet been fetched. The order word register, in the

* Blocks of memory addresses assigned to program stores and call stores are fixed for all No. 1 ESS installations; the wiring pattern of the memory address decoder is therefore the same in all installations.

† BB is a symbolic representation of some constant; AA represents a program address.

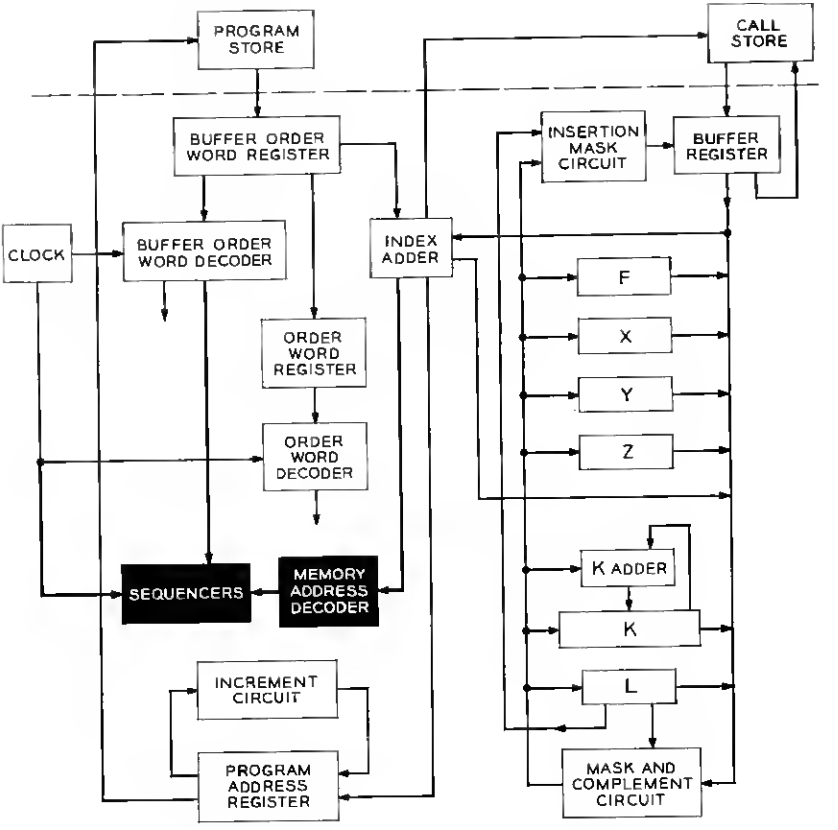


Fig. 14 — Derivation of the central control (8).

meantime, has the instruction at address AA, while the program store address register has received a data address from the index adder. This data address Y + BB is now being used to read the program store. At time 3 the buffer order word register contains the information located at address Y + BB, and the order word register continues to hold the instruction at address AA, while the program store address register has now been incremented to the value AA + 1 to prepare for the reading of the next instruction. At time slot 4 this instruction has been read into the buffer order word register, the output of the buffer order word register has gone via the index adder to the appropriate destination (the X register) under the control of the order word register, and the program address register is preparing to read the instruction at address

TIME SLOT	BUFFER ORDER WORD REGISTER	ORDER WORD REGISTER	PROGRAM ADDRESS REGISTER
1	(AA)	(AA − 1)	AA + 1
2*	[AA + 1]	(AA)	BB + Y
3*	(BB + Y)	(AA)	AA + 1
4	(AA + 1)	(AA)	AA + 2
5	(AA + 2)	(AA + 1)	AA + 3

(—)→ SYMBOL MEANING WORD STORED AT THIS ADDRESS OR IN THIS REGISTER.
INSTRUCTION AT ADDRESS AA IS MX, BB, Y;
BB + Y IS AN ADDRESS OF DATA IN THE PROGRAM STORE.

*THESE ACTIONS ARE CONTROLLED BY CENTRAL CONTROL INTERNAL SEQUENCE CIRCUITS SINCE THE OPERATION COVERS MORE THAN ONE CENTRAL CONTROL CYCLE TIME.

Fig. 15 — Time sequence of words passing through BOWR, OWR, and PAR when reading data from program store.

AA + 2. At time 5 this instruction is in the buffer order word register and the order word register has the instruction located at address AA + 1, while the program store address register has been incremented to AA + 3. Since the order word decoder is strictly a combinational circuit, the sequencer must be used to control the actions of fetching the data; otherwise the order word register would simply control the execution of the instruction at address AA three consecutive times. Note that the instruction for reading data from the program store consumes three cycles: one basic cycle, one cycle to read the data from the program store, and one cycle to reread the next instruction.

3.13 *Conditional Transfers*

A very important part of any data processing machine instruction repertoire is the set of conditional transfer instructions. These instructions cause a transfer of program control to a specified address if some data word or bit of data appearing in central control is some predetermined value. If the word or bit does not have that value, the transfer is not made, and the instruction immediately following the transfer instruction is executed.

Eight transfer instructions are provided to interrogate the contents of the accumulator for the following values: positive, negative, arithmetic zero,* all but arithmetic zero, logical zero (+0 only), all but logical zero, less than or equal to arithmetic zero, and greater than or equal to arithmetic zero.

* Arithmetic zero includes +0 (all zeros) and −0 (all ones). In both cases all 23 bits are alike, and arithmetic zero is therefore also referred to as "homogeneity."

A pair of control (C) flip-flops connected to the masked bus (see Fig. 16) store the homogeneity and sign of data words read from memory as the words appear on the masked bus. Another eight transfer orders test the C flip-flops for the same combinations of values available for testing the accumulator register.

Normally, the conditional transfer instructions follow immediately after the condition is registered, either in the accumulator or the C flip-flops. The usual instructions for gating information into registers may set the accumulator or the C flip-flops. In addition, there is a set of compare instructions (see Fig. 17) which do not alter any register but

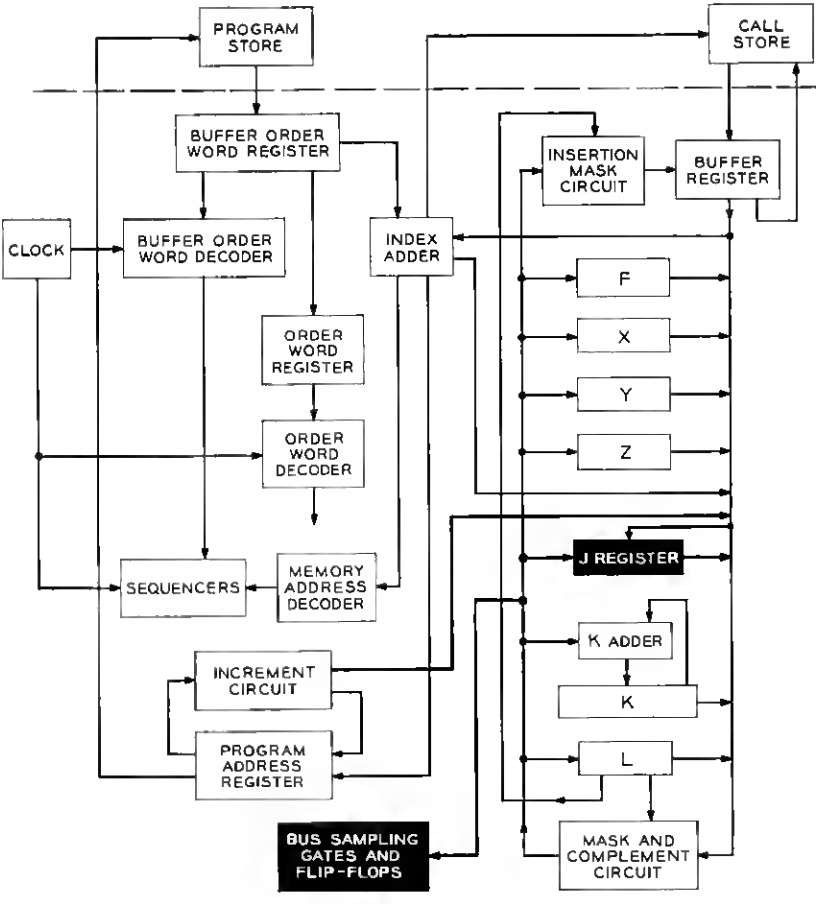


Fig. 16 — Derivation of central control (9).

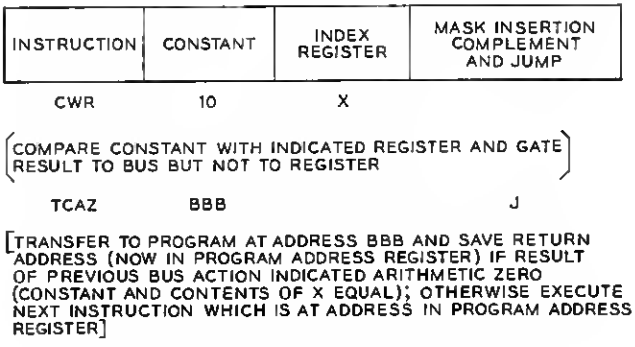


Fig. 17 — Basic instruction format 4.

which set the C flip-flops according to the result of the comparison. For example, the instruction CWR,10,X compares 10 with X and places the result in the C flip-flops. The comparison is performed by subtracting 10 from X, placing the result on the bus, and gating it only to the C flip-flops. One way to check whether the X register was actually equal to 10 is to follow the first instruction with an instruction TCAZ, BBB: transfer to the address BBB if the C flip-flops are equal to arithmetic 0; if not, advance to the next instruction.

3.14 *J Option and Register*

Associated with transfer instructions is a return address (jump) option (see Fig. 17). Unconditional or conditional transfer instructions occur frequently in the middle of a program and are used to transfer to a subroutine to do a common task; subsequently the subroutine returns control to the original program. Since the subroutine must know where to return, a J register (see Fig. 16) has been provided which may be set up at the discretion of the programmer whenever a transfer is executed. If the transfer is actually executed, the J register is set to the address of the instruction immediately following the transfer instruction. To set up the J register, a path must be provided from the output of the increment circuit to the unmasked bus and thence to the J register (see Fig. 16).

3.15 *Index Register Modification Options*

Index register modificatiou options are available in the No. 1 ESS order structure. If some task is being performed on a number of successive memory locations, it is sometimes convenient to set an index

register to the value of the first memory address, then to increment the register by +1 as successive words are read from memory. This incrementing can be performed as an option on most reading instructions. For example, the instruction MX,3,Y simply gates the contents of memory at the address Y + 3 into the X register. The instruction MX,3,YA (see Fig. 18) gates the contents of memory at the address Y + 3 into the X register and increments Y by 1. The incrementing is performed by connecting the increment circuit input to the unmasked bus (see Fig. 19); this allows the index register to be gated into the increment circuit; the output of the increment circuit may later be gated to the index register via an output connection to the masked bus.

Two other index register modification options exist which change the indicated index register to the indexed quantity. Thus, the instruction MX,3,YW reads the memory at address Y + 3 and gates this into the X register and also changes Y to the value W, the indexed quantity, which is Y + 3. Another index register modification option is the setup index register modification. For example, MX,30000,SY would read the contents of memory at address 30,000 into the X register and would place the quantity 30,000 into the Y register (SY = set up Y).

3.16 *Logic Register Setup Options*

The logic register is changed very frequently in the course of a typical program. Furthermore, many of the programs that are encountered include effectively indirectly addressed readings,* i.e., one instruction is used to read a quantity, and the reading is then used as the address of the quantity which is actually desired. In such a case, the register being used for the second reading usually contains the complete address so that no additional data are required as the constant of the instruction. Consequently, it is desirable to be able to use the constant of the second instruction to set up the logic register instead of performing unnecessary indexing. A direct path has therefore been provided between the buffer order word register and the logic register (see Fig. 19). If setup masking is specified, then the constant of the instruction is used to set up the logic register and is not used in indexing.

The interplay of index register modification and setup masking options can be illustrated by the following programming problem: complement the 3 least significant bits of memory at the address Y + 3.

* Indirect addressing was not considered worthwhile for data operations, because it never saves time and rarely saves instructions in this system.

INSTRUCTION	CONSTANT	INDEX REGISTER AND MODIFICATION	MASK INSERTION COMPLEMENT AND JUMP
MX	3	YA	

[READ CONTENTS OF MEMORY AT ADDRESS Y+3 INTO X REGISTER; THEN INCREMENT Y BY +1]

Fig. 18 — Basic instruction format 5.

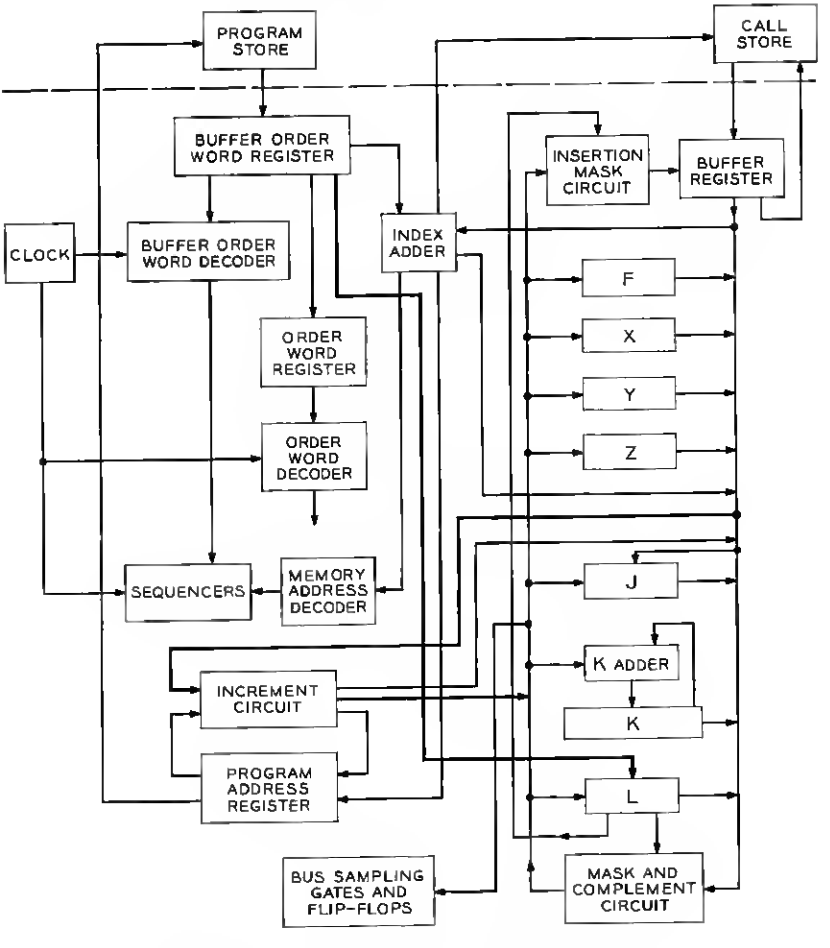


Fig. 19 — Derivation of the central control (10).

Using straightforward programming, this can be done by the following three-step program: MX,3,Y (read memory into the X register); WL,7 (set up the logic register to the value seven, i.e., 1's in the 3 least significant bits, 0's elsewhere); XM,3,Y,ELC (store X in memory using insertion masking and complementing at the address Y + 3). This program will result in 3 least significant bits of X — i.e., the bits selected by the logic register — being complemented (C option) and inserted (EL option) into, first the buffer register, thence the memory, at the location Y + 3. A two-step program for performing the same task is the following: MX,3,YW (read from memory at address Y + 3 into the X register and set up Y to the value Y + 3); XM,7,Y,ESC (set up the logic register to the quantity 7 as indicated by the S in the ESC mask option; complement, as indicated by C, the X register, and insert, as indicated by E, into the buffer register the 3 least significant bits as selected by the L register; and gate this information to memory at the address which is now in Y, and which is 3 greater than the original value of Y). Note that the first program repeated the constant 3 twice, whereas the second program used it only once. The index register modification option permitted the constant of the instruction to be remembered for subsequent instructions without using any extra steps. Similarly, the constant 7 for setting up the mask was useful in the second instruction of the modified program because, since no constant was necessary for addressing the memory, a constant could be used for setting up the logic register.

It is important to remember that these options not only conserve memory space for instructions but save the time necessary to execute additional instructions. In this system, each instruction takes one cycle whether it be a memory instruction (M) or a register setup instruction (W).

3.17 *Rightmost One Function*

One function that occurs frequently in the type of data processing work that constitutes the call processing program of the No. 1 ESS is that of detecting and identifying a one in the midst of a group of zeros in a word. The one might signify a request for action, the zeros, inactivity; the position of the one would represent which member of the group requires the action. By concentrating always on the least significant one, successively all requests will eventually be handled.

It is important to have some instruction which identifies the position within a word of the rightmost one, because this operation is performed often and is awkward to do using more conventional instructions. The

word to be examined is placed in the accumulator; a rightmost one detect circuit connected to the accumulator gates the binary position of the rightmost one onto the unmasked bus. This information is then transmitted via the buses to the F (for "first-one") register. A circuit to reset this bit in the accumulator then receives its selection information from the F register.

Two instructions exist for the first-one function: TZRFZ transfers if the accumulator is zero; otherwise, it gates the position of the rightmost one to the F register and resets that bit; TZRFU performs the same actions, except that the bit in K is not reset. The programmer specifies the transfer address information in the same way that it is specified for any conditional transfer instruction.

3.18 *Buffer Bus Registers*

In addition to the index registers described above, a number of other flip-flops in the central control are under the control of a programmer. They include the bulk of the maintenance control and match flip-flops. These flip-flops are in groups and are set up and read by buses connected to the B register (see Fig. 20). The flip-flop groups are then examined and controlled as if they were word locations in memory. Each flip-flop group is assigned a distinct address; when this address is generated, the memory address decoder operates the gates to or from this flip-flop group. Thus the control of registers does not consume instruction code space.

3.19 *Interrupt Facilities*

The No. 1 ESS central control has interrupt facilities. These facilities permit a signal to come in at any time and:

(1) cause the program currently being executed to be interrupted,

(2) permit the state of central control to be stored in memory,

(3) allow an interrupt program to be executed,

(4) allow the state of the central control to be restored to its pre-interrupt state, and

(5) allow the interrupted program to be resumed.

In effect, the programmer need not be concerned about the possibility of an interrupt occurring at any time, since the interrupt will not interfere with the execution of his program.

The interrupt is used for two purposes: first, dial pulse scanning and similar functions which must be performed every five or ten milliseconds are carried out by an interrupt program, triggered by a five-millisecond

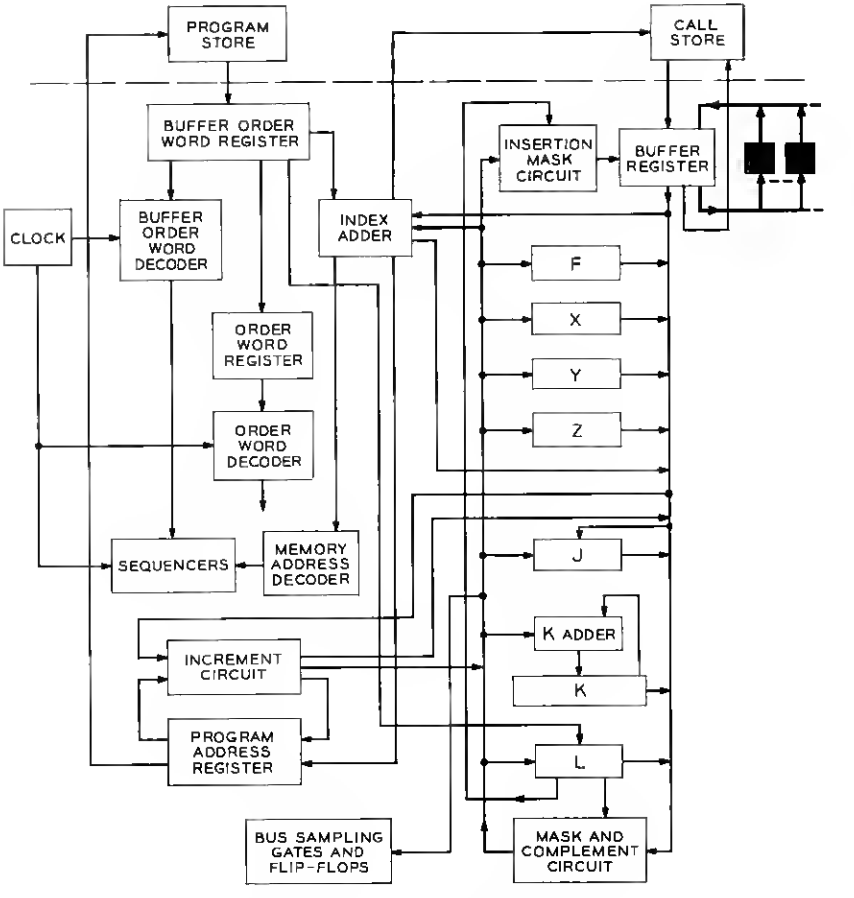


Fig. 20 — Derivation of the central control (11).

clock within the central control. Second, any trouble indication leads to an immediate interrupt to analyze the trouble and take corrective action before the trouble source causes errors in calls.

Call processing programs are carried out at the base level, which may be interrupted by any of the interrupt sources. Several levels of interrupt exist. A signal from a higher level has the ability to interrupt a program initiated by a lower-level interrupt. Most of the special interrupt facilities are obtained through the use of the interrupt sequencer.

In general, there is no problem of interaction among different interrupt programs and between the base-level program and interrupt programs, *provided they do not both write in the same sections of memory.*

Sometimes, this is unavoidable; for example, the program that detects an incoming trunk service request (an interrupt-level program) and the program that seizes an outgoing trunk on a terminating call (a base-level program) both write into the same busy-idle bits if the trunk is a two-way trunk used for both incoming and outgoing calls. Interaction problems may occur if a bit is being inserted in memory and the interrupt occurs between the reading and writing steps that constitute an insertion into memory. Two instructions have been created to solve this problem: MCII and MKII. These instructions are the normal MC (memory to the buffer and also the C flip-flops) and MK, except that all but the maintenance interrupts are barred until the immediately following instruction has been executed. By using one of these instructions as the first step of an insertion, a programmer guarantees that no interfering interrupt will occur while he is inserting the desired information.

3.20 *Mixed Indexing*

Sometimes, a couplet of instructions such as

MX,0,Y

MZ,0,X

occurs. The second of these instructions uses the value of X that was set up by the first instruction. Since the call store reading of the first instruction comes back at the same time that indexing is performed for the second instruction, a timing problem exists. This is handled by recognizing such situations (mixed indexing) and gating the data on the masked bus, i.e., the data going into the X register, to the index adder (see Fig. 20), instead of gating data from the unmasked bus, i.e., the present contents of the X register. The circuit for recognizing this situation must examine the buffer order word register and the order word register to check for this condition. Note that if an interrupt takes place between these two instructions, then the X register will have been set up to the new value, and no mixed indexing takes place; in effect, the second member of the couplet has been preceded by a vacant or no-op instruction.

3.21 *Early Transfer Instructions*

For certain highly repetitive programs, especially those involving scanning, it is desirable to have a conditional transfer instruction which

will consume additional cycles only if the transfer is made. This is accomplished by coupling a normal read or write instruction with an indication that a transfer to a preset address is to be made if the C flip-flops or the K register records a particular state. The instruction is called an "early transfer" instruction because, if the transfer is made, the reading or writing action is inhibited; the transfer decision must be made sufficiently early so that the unwanted action is prevented.

Two of the early transfer instructions are TCMMF and TAULM. TCMMF will transfer to the address (previously set up) stored in the J register if the C sign flip-flop shows a 1 (or minus); otherwise, a normal MF (memory to the F register) instruction is executed. All options normally available for an MF instruction can be specified, since the TCMMF operation itself completely specifies the conditional transfer instruction and does so without permitting any options or any choice of the source of the transfer address. TAULM will transfer to the address (previously set up) in the Z register if the C flip-flops show a nonzero quantity; otherwise, a normal LM (logic register to memory) instruction is executed. Again, all options normally available for an LM instruction can be specified; the TAULM operation completely specifies the conditional transfer instruction and transfer address source.

The advantage of the early transfer instructions is that the transfer address need be set up only once for a large number of loops of a subroutine, or that the transfer address may have been previously set up in the course of executing another part of the program; if no transfer takes place, no cycles have been wasted on making the decision.

3.22 *Logical Combinations of Registers with Memory or Data*

Instructions are available which permit indexed data (W) or the contents of memory found at an indexed location (M) to be logically combined with the contents of the X, Y, or Z register. This is accomplished by first gating the X, Y, or Z register to the L register via the buses, then using the mask and complement circuit to logically combine L with either M or W and gate the result back into X, Y, or Z. AND, OR, AND complemented, and OR complemented are the logical expressions that may be specified; naturally, this means that the mask circuit must be able to OR as well as AND. Since these instructions use the L register, no masking may be specified.

IV. PERIPHERAL SYSTEM FACILITIES IN CENTRAL CONTROL

In addition to communicating with the stores, the central control also communicates with the peripheral system.[12] This system contains three

main bus systems (Fig. 21). The first of these is used in communicating with the central pulse distributor.[3] This central pulse distributor either operates flip-flops which drive relays directly or is used for selecting the particular unit which is to be addressed via the second bus system, the peripheral address bus. Responses from peripheral units come via the scanner answer bus.

Peripheral actions are not generally performed in the middle of a complicated data processing problem. Therefore, some of the general purpose index registers of central control are used for driving these buses (Fig. 22). The F register is used to drive the translator which controls the central pulse distributor. The logic register is used to receive answers from the scanner response bus. This makes it easy to combine scanner answers with memory information, since the memory information can be read into the buffer register and can be combined logically with the contents of the logic register in the mask and complement circuit.

Ideally, it would be most reasonable to drive the peripheral address bus from the data buffer register. However, the timing of the waves of information leaving central control to address the peripheral system is such that the buffer register would not be available for a reading on the subsequent cycle; peripheral actions start after a considerable amount of preliminary processing, frequently including a call store reading. For this reason the contents of the buffer register are transmitted to the accumulator addend register, which is known to be available at this time. The accumulator addend register is used to drive the translators connected to the peripheral addressing bus.

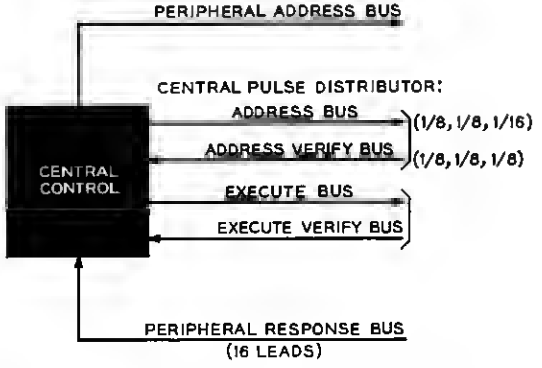


Fig. 21 — Block diagram of central control communication with peripheral equipment.

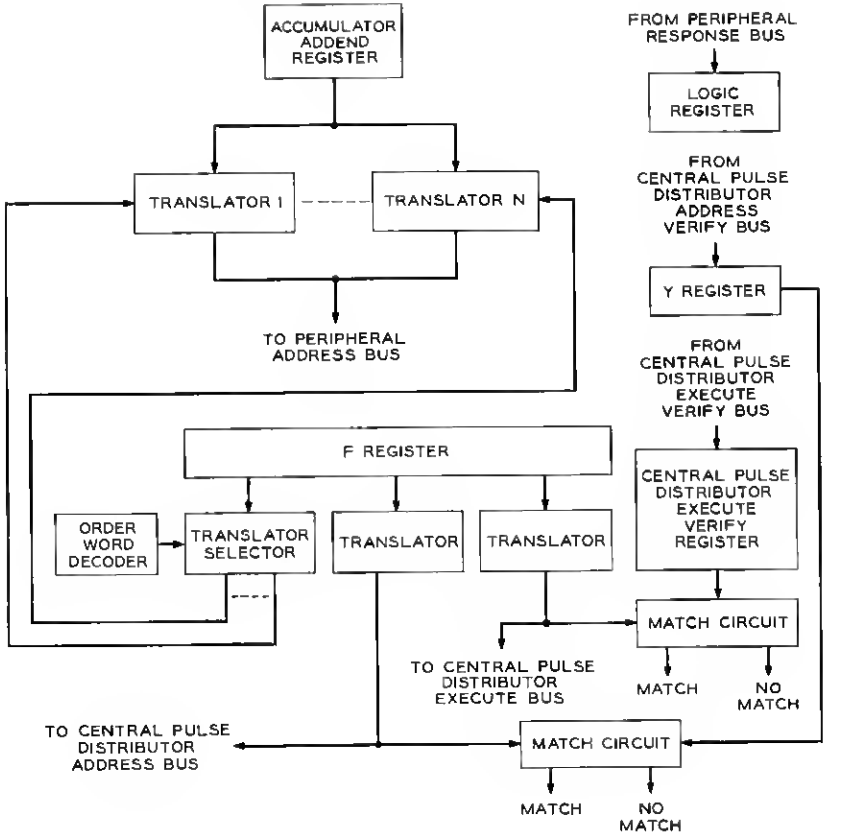


Fig. 22 — Detached internal central control blocks for communication with peripheral equipment.

A large majority of the units driven by the peripheral system are network units,[4] signal distributors,[3] and scanners.[3] For economic design of the many peripheral controllers, a coded address is used for controlling these units. For example, a 1024-point scanner requiring a 1-out-of-64 row selection is addressed by two 1-out-of-8 signals, not by 6 binary signals. Furthermore, the address code for each of the various network frames and the signal distributors is different. Central control has built within it translators to convert the binary information, convenient for data processing, to the particular code used for controlling a specific unit.

Since many peripheral instructions are sent out from a section of call store called the "peripheral order buffer," it is desirable that the actual

peripheral instructions be independent of which particular type of frame is being addressed. Since the enable address for a particular frame is always required along with the peripheral instruction, it is convenient to store the type of frame along with this enable address so that the proper translation of the binary addressing information will be made. A peripheral action is therefore preceded by setting up the F register to the enable address necessary for selecting the particular unit and a code which will select the proper translation option of the binary information to be sent to the peripheral bus. The actual peripheral instruction is then set out via an instruction, such as MA, which assumes that the F register has been preset. MA first sends out an enable signal, then takes the contents of memory and gates them via the buffer register and accumulator addend register to the peripheral address bus, translating the binary information according to the code stored in the F register.

When a peripheral (scanner) response is to be returned to central control, it is sent to the logic register. Therefore the instructions for controlling scanners are different from the instructions for simply sending an instruction to a peripheral unit which does not give a response. The basic general-purpose peripheral instructions are therefore MA and MAS (MAS being used when a response is expected) and the counterparts, WA and WAS. The MAS instruction and WAS instruction also reset the logic register and open gates to this register from the peripheral response bus.

If a central pulse distributor is addressed only for the purpose of operating or resetting a flip-flop, a special instruction, MD or WD, is used. With these instructions it is only necessary to set up the F register, since the peripheral addressing bus is not required.

For addressing the scanner, only 6 bits of information are necessary to select a row. There is enough information in a 23-bit word to set up an enable address and to actually address the scanner. The MSF and WSF instructions, by simultaneously setting up the F register and the accumulator addend and then performing a peripheral operation, accomplish this in one setup cycle. (The actual execution of the instruction overlaps into the next cycle so that two scanner readings cannot be made in consecutive cycles.)

For purposes of verifying peripheral operations, a verify response signal comes back from the central pulse distributor. This response is gated into the Y register and is then matched against the output of the translator that is connected to the F register. (No response is returned when executing MD or WD instructions.)

For driving such units as the tape unit,[5] which will accept straight binary information, and for sending test signals to peripheral units, it is also possible to bypass the translator, simply letting the outputs of the accumulator addend register go directly to the addressing bus. Since the peripheral addressing bus is 36 bits wide, it is possible, if testing a unit which requires more than 23 input leads, to take 13 bits from the accumulator as well as 23 bits from the accumulator addend.

It is important to remember that in the case of network frames the peripheral addressing bus contains instructions as well as addressing data.[4]

Another check made on peripheral operations is that the proper central pulse distributor has been selected. When a central pulse distributor is selected, it sends back an echo signal which goes to a flip-flop register on the buffer bus. This is matched against the output of a translator attached to the F register.

V. SUMMARY OF ESS ORDER STRUCTURE WITH OPTIONS

This section is a summary of No. 1 ESS instructions with their available options, as shown in Table I. As has been previously indicated, every instruction has three main modifiers: the data field; the RM field, which includes index register modification and indirect addressing for transfers; and the LCJ field, which includes masking options, complementing, and the setting up of the J register on transfers. In addition, although this is not specified in the actual writing of each instruction, many instructions set up the C flip-flops, which can then be examined on a subsequent TC conditional transfer instruction.

The constant in the data field may serve one of three purposes: it may be directly used data that is part of the instruction; it may be part of the address used for finding such data; or it may be the mask that is to be used in the instruction. In Table I, the symbol S in the data or address section indicates that data or an address may be specified unless an S occurs in the LCJ field, in which case no data or address constant may be specified, since the constant of the instruction must be used for setting up the mask.

The R subfield usually contains the identity of the indexing register; in a few instructions, the R subfield specifies a register indicated by the instruction. The latter include the instructions for adding the contents of two registers (since only one of the registers can be specified as part of the instruction, the other register must be specified as an option); the CWR instruction, which is used to compare a register with a constant in the order; and the TR family of instructions which sense the

TABLE I — SUMMARY OF BASIC ORDERS AND AVAILABLE OPTIONS

Symbolic Order Fields	DA		RM						LCJ						
Subfields	D	A	R*					M	L†				CJ		
General-purpose operation codes	Data	Address	Identity of R in OP Code	Indexing Register	A (R + 1 → R)	W (DA + R → R)	S (DA → R)	Indirect Addressing	PL Masking	EL Masking	PS Masking	ES Masking	C Complementing	J (RA → J)	Set C Flip-Flops
WK, AWK, SWK, PWK, UWK, XWK	S			√	√				√		√		√		
WF, WJ, WX, WY, WZ, CWK, CWKU	S			√	√				√		√		√		√
WL, PWX, PWY, PWZ, UWX, UWY, UWZ	√			√	√								√		√
WB	S			√	√				√	√	√	√	√		√
H, HC, Q, QC	√			√	√										
MC, MCII		S		√	√	S	S		√		√		√		√
CWR	S		√						√		√				√
MB		S		√	√	S	S				√				
BM		S		√	√	S	S					√			
ABR, AFR, AJR, AKR, ALR, AXR, AYR, AZR	L		√						√		√		√		√
SBR, SFR, SJR, SKR, SLR, SXR, SYR, SZR	L		√						√		√		√		√
FM, JM, KM, XM, YM, ZM,		S		√	√	S	S		√	√	√	√	√		
MK, AMK, SMK, PMK, UMK, XMK, MKII		S		√	√	S	S		√		√		√		
MF, MJ, MX, MY, MZ, CMK		S		√	√	S	S		√		√		√		√
LM		S		√	√	S	S			√	√	√	√		
ML, PMX, PMY, PMZ, UMX, UMY, UMZ		√		√	√	√	√						√		√

TABLE I — *Continued*

Symbolic Order Fields	DA		RM						LCJ						
Subfields	D	A	R*					M	L†				CJ		
General-purpose operation codes	Data	Address	Identity of R in OP Code	Indexing Register	A (R + 1 → R)	W (DA + R → R)	S (DA → R)	Indirect Addressing	PL Masking	EL Masking	PS Marking	ES Masking	C Complementing	J (RA → J)	Set C Flip-Flops
T, TK... TC...		√		√	C	C	C	√						√	
TR...		√	√		C		C	√	√				√		
Input-output operation codes															
MA, MAS, MSF, MD		S		√	√	S	S		√		√		√		
WA, WAS, WSF, WD	S			√	√				√		√		√		
Combined operation codes															
TZRFU, TZRFZ		√		√	√	√		√						√	
TAULM, TCMMF		+ S		+ √	+ √	+ S	+ S		+ √		+ √		+ √		

* Of options A, S, and W, only one may occur in any one instruction.
† Not more than one of the four L options can be used in an instruction.
Key to symbols:
√ — indicated use of field is available
S — √ unless S appears in L subfield
C — conditional and late: occurs only if transfer occurs and after register is used (or, in the case of TR... orders, after the register is tested)
L — √ only if S appears in L subfield
\+ — action of options occurs only if transfer does not occur.

contents of some register and transfer accordingly. Three index register modifications are available, of which only one, the A option, is available if setup masking is used. (This is a direct result of the fact that only one constant may be specified in any instruction; if this constant is used to set up the mask, then it cannot be used to modify an index register.)

Some general observations may be made concerning the types of options available with various instructions. W instructions do not have W or S index register modification options. While these index modification options are meaningful, they would bave a relatively low utility and require a great deal of code space. The W and S options are much more useful on memory instructions for setting up a register to a full

address, so that on a subsequent instruction, the constant in the data field will be available for masking.

Certain instructions, such as PWX, UMY, WL, ML, etc., do not permit setup masking, since they either set up the mask directly as part of the instruction (WL), or the logic register is used in carrying out the instructions (UMX, etc.).

On MB instructions, no masking or complementing is possible, since the contents of the buffer register are fed directly by the memory and do not pass over the bus. The PS option is included and allows a programmer to set up the logic register for a subsequent instruction even though he does not use it in this instruction.

Insertion masking is used only on WB instructions and on the instructions which write the contents of some register to memory, since the insertion mask can only be associated with the B register.

The C flip-flops in general are set up on all compares and on all orders which gate W or M to some register other than K.

As can be seen, the rules for checking on which options are allowed on any particular instruction are not too simple. Therefore, even though they are summarized in Table I, the No. 1 ESS program compiler[13] checks for violations of the allowed options. There are, of course, a number of other restrictions that the compiler can check for. The chief restriction within No. 1 ESS is the fact that the accumulator may not be used as the indexing register on the instruction immediately following AMK, PMK, UMK, XMK, or SMK instructions. Following all peripheral instructions, there are a number of very complicated restrictions; in general, the Y and F registers cannot be altered on the next instruction, and the L register cannot be changed following MAS, WAS, MSF, and WSF.

VI. ENCODING OF THE ORDER STRUCTURE

Each instruction or program order word obtained by central control contains 37 information bits designated 36, 35, $\cdots$, 0. Each such order consists of an order field and a data-address field. The order field includes an order selection subfield, an index register subfield, and order option subfields. When the order includes a data word, the data-address field contains the data word in bits 22 through 0, and bits 36 through 23 compose the order field. When the order word contains an address in the data-address field, the address occupies only bits 20 through 0 of the word, and a larger order field appears in bits 36 through 21 of the program order word.

The 21-bit address field permits full access by memory reading and

writing orders to all locations in the program stores and call stores, as well as to a group of flip-flop registers in central control. When an order option subfield indicates the setting up of the logic register (e.g., PS or ES masking), the data-address field contains the 23-bit word of data; in such instances, the order field is restricted to bits 36 through 23, even when the order selection subfield indicates a memory reading or writing order.

The encoding of the instruction repertoire or order structure represents a compromise between: (1) attempting to provide an order structure with the maximum flexibility that can be represented by the available binary combinations and (2) decomposing the combinations of order selection, index register selection, and option selection into simple subfields. Counting all meaningful combinations of order selection, index register selection, and nonconflicting order option values, the encoding provides over 12,000 distinct combinations in the 14- and 16-bit order word.

The index register subfield is always encoded in bit positions 34 through 32. Bit position 35 is reserved for the complement option for all orders except regular transfer orders; bit position 35 serves as a J option subfield for transfer orders. A complete decomposition is not possible for index register modification options or masking options without an excessive waste of code space. However, the encoding includes a grouping of the classes of related orders, this grouping being represented by relatively simple bit combinations; within each grouping the index register modification and masking options are each grouped into one-, two-, or three-bit subfields according to the number of meaningful and useful combinations.

The encoding is shown as the Karnaugh maps in Figs. 23 through 26. These maps represent the four binary combinations of bits 31 and 30; this division of the encoding is representative of the grouping of several large classes of orders. For example, the binary combination 31-30 = 01 is assigned exclusively to orders reading memory; all such orders are encoded within this combination and its corresponding map in Fig. 23. The combination 31-30 = 11 is assigned to W orders in which the destination register is given explicitly by the mnemonic code and not by the index register field ($R_D \neq R_I$); the encoding of this class is shown in Fig. 24. Related orders, such as MA and WA, occupy corresponding positions within the two maps. This correspondence provides for economy in designing the gating functions which carry out the same steps for related orders. For example, a single destination register selector can be activated by either a memory reading order or its corresponding W order.

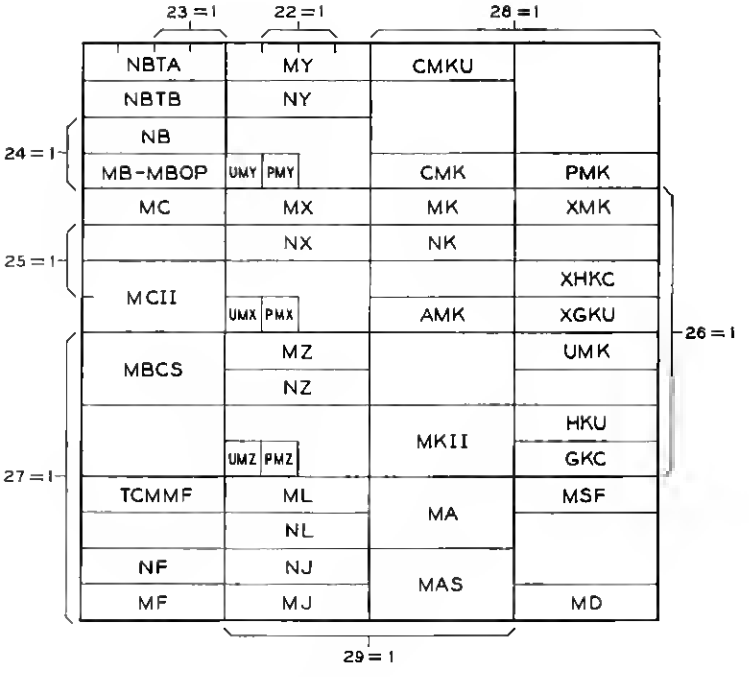


Fig. 23 — Memory reading orders (31-30 = 01).

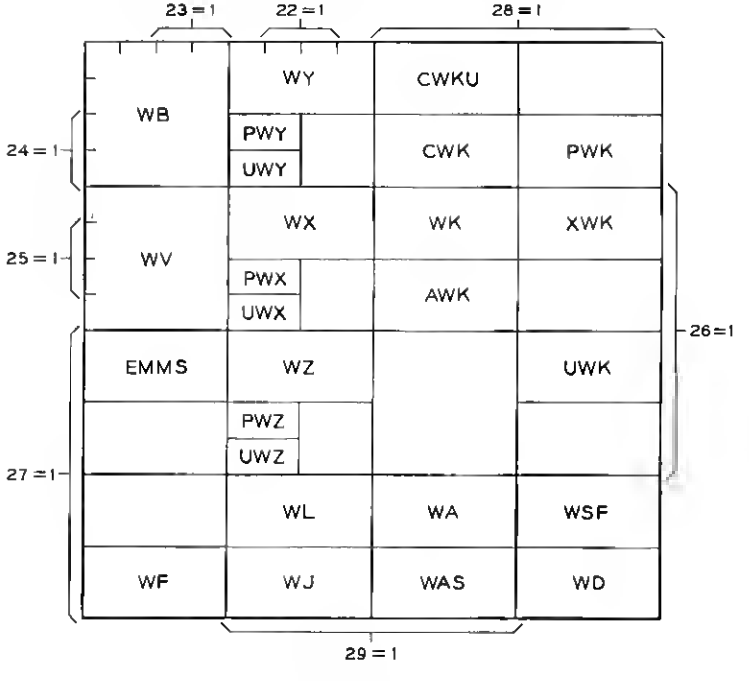


Fig. 24 — Word orders ($R_D \neq R_I$); (31-30 = 11).

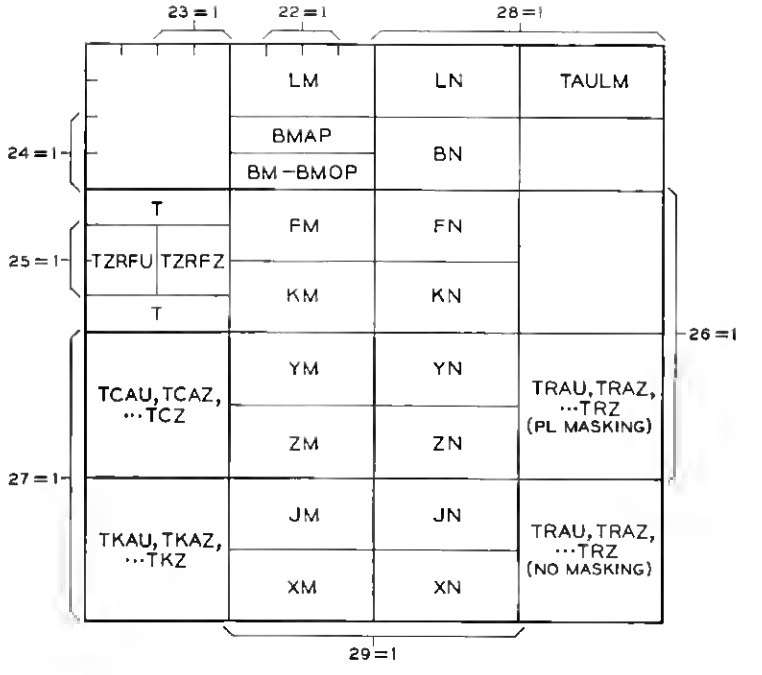


Fig. 25 — Writing orders and regular transfer orders (31-30 = 00).

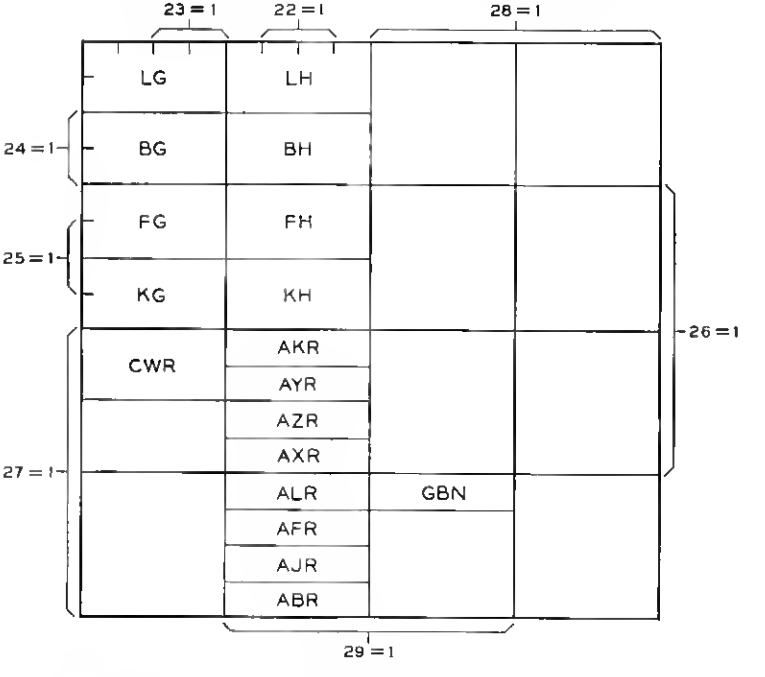


Fig. 26 — Maintenance writing orders and word orders ($R_D = R_I$); (31-30 = 10).

The remaining classes of orders require less coding space than the two classes just described and consequently occupy only portions of the remaining maps in Figs. 25 and 26. The combination 31-30 = 00 includes all regular transfer orders, all but a special class of memory writing orders, and a small class of miscellaneous orders including rotate and shift orders. Fig. 26 represents the binary combination 31-30 = 10 and includes the class of maintenance writing orders and the word orders ($R_D = R_I$). Figs. 23 through 26 include special classes or orders described below. The maintenance writing orders include control mode memory orders (represented by the letter N in the mnemonic equivalents), G-mode memory orders, and H-mode memory orders (see Section 7.7).

The early transfer orders are encoded according to the data processing actions to be taken whenever the decision is made not to transfer. Thus the order TCMMF is encoded as a memory reading order (31-30 = 01), and the order TAULM is encoded as a memory writing order.

With the encoding just shown, the decoders were designed to optimize whatever decompositions are made available for both the optional data processing gates and in the selectors and gates common to related orders. In addition to the buffer order word decoder and the order word decoder, two classes of data processing functions must be included to complete the above summary. The memory address decoder controls the generation, transmission, and central control response of each program store command and each call store command; the data processing for memory reading orders, memory writing orders, and indirect transfer orders includes the use of the memory address decoder. The decoders described here carry out those gating actions necessary to obtain and process single-cycle orders. Many classes of orders cannot be executed in a single machine cycle; the additional gating actions for such orders are handled by a group of sequencers; these actions may include automatic retrial and/or correction of program order words and data words.

VII. MAINTENANCE OBJECTIVES AND CIRCUITS

The No. 1 ESS must be able to provide continuing service to customer lines in the face of occasional and random occurrences of circuit troubles.[14] Duplication of subsystem units or portions of such units provides a set of potential replacement parts. Whenever a circuit trouble occurs within a subsystem, the detection of that trouble is followed by the required replacement; this replacement is made at electronic speeds to minimize the interfering effect of the trouble.

Each subsystem includes maintenance circuits which: (1) serve in detecting symptoms of circuit troubles as they appear and (2) aid in determining the location of such trouble to facilitate repairs

The detection of circuit troubles leads to the execution of maintenance programs which first determine whether a fault exists and if so whether the circuit trouble occurred within the active (controlling) switching system or within a standby duplicate unit. If a fault has occurred in an active subsystem unit, the next step is the necessary switching of associated active and standby units. The system is therefore quickly restored to an operable state and returned to the normal business of processing telephone calls; the subsystem unit in trouble is placed in an out-of-service state; and finally, special program sequences are interleaved with call processing programs to determine the faulty circuit element. The maintenance actions last described constitute a diagnosis of the out-of-service unit by the switching system; the results of this diagnosis appear as a printout on a special teletypewriter unit. Corresponding to each such printout is an entry in a specially prepared dictionary[14] which the maintenance man consults; the "definitions" in this dictionary are a listing of plug-in circuit packs to be replaced.

The entire procedure just described, from the detection of a circuit fault to the replacement of associated circuit packs, takes place in a matter of a few minutes; on completion of the repair of the out-of-service subsystem unit it is returned to the standby state for protection against future occurrences of circuit troubles. The maintenance circuits and associated program sequences serve in meeting a primary maintenance objective: essentially continuous telephone service with a minimum degradation in the quality of service in the presence of occasional circuit troubles.

Certain of the maintenance actions operate continuously and independently of the program sequences being executed in the central processor; other actions are obtained with the maintenance circuits and special program sequences. The integration of such program sequences and maintenance circuits is described elsewhere in this issue.[14]

7.1 *Circuit Checks of Communication Channels between Central Control and Connecting Subsystem Units*

Communication of information between central control and the remaining subsystems comprises the transmission of commands and addresses to one or more such units; each command specifies the required circuit response, and each address specifies the location or locations

which are to respond to the command. These responses include setting (resetting) flip-flops, scanning a group of ferrods, reading or writing a 24-bit word of call store data, etc. Accordingly, the circuit responses in some instances include the return of information to the subsystem unit generating the command and address. Circuit check signals accompany redundantly encoded commands, addresses and responses transmitted between the central control and (1) the program stores, (2) the call stores, (3) the central pulse distributors, and (4) peripheral units such as scanners, network controllers, and signal distributors. Maintenance circuits in both central control and the connecting units provide a continual check of communication and serve as a safeguard against noise and circuit troubles in the communication channels. Since the rate of communication between central control and its connecting units is quite high, most occurrences of circuit troubles within (1) the circuit generating the redundancy in the commands, addresses, and responses; (2) the registers for transmitting and receiving these commands, addresses, and responses; and (3) the associated check circuits, will quickly result in the detection of a check failure. For example, the check of communication between central control and the program store is briefly outlined in Fig. 27. The decoder combines clock signals from the microsecond clock with dc inputs to generate synchronizing, command and mode signals. These are transmitted along with a 16-bit address and four-bit code to select the appropriate program store information block via cable drivers connecting a twisted-pair cable leading to the program stores. The selected program store responds with synchronizing and check signals and a 44-bit reading from the twistor memory which is returned to the buffer order word register of central control. As this response is being returned, the contents of the program address register are transmitted to an auxiliary storage register so that the next information block code and address may be gated to the program address register. This last gating action permits the simultaneous check of one program store response and the generation and transmission of a following program store command. Under control of timing signals derived from the decoders, the check circuits carry out single-error and double-error checks of the information contained in the auxiliary storage register (corresponding to the address of the word obtained)[15] and the buffer order word register. The check circuit also verifies that the program store response included a check (all-seems-well) signal from the responding program store. When one or more of these checks fail, signals on the corresponding check-fail conductors lead to the required remedial circuit action.

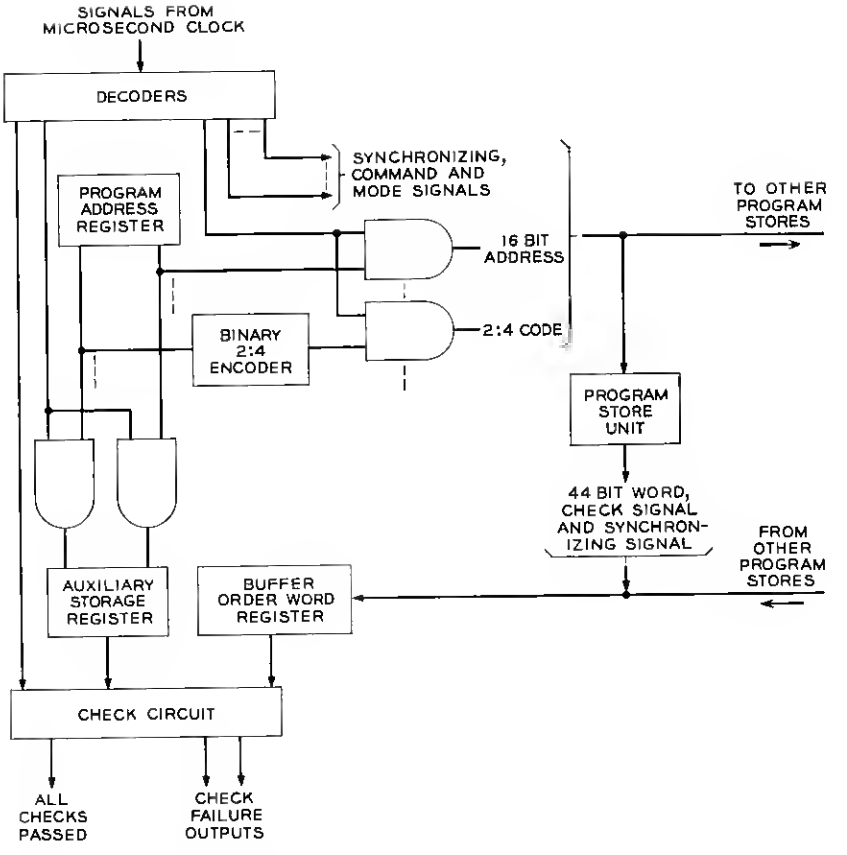


Fig. 27 — Check of central control-program store communications.

7.2 *Interrupt Actions*

A simplified block diagram of the interrupt system is shown in Fig. 28; it includes the three flip-flop registers in central control that are part of the buffer register bus system. This access permits single-cycle reading or writing access to these registers similar to that available to call store memory locations.

The interrupt source register comprises a number of interrupt source flip-flops; input signals to this register arrive from the millisecond clock and various check circuits within central control. The interrupt-level activity register serves to record the level interrupt corresponding to the program sequences being executed in central control. That is, cor-

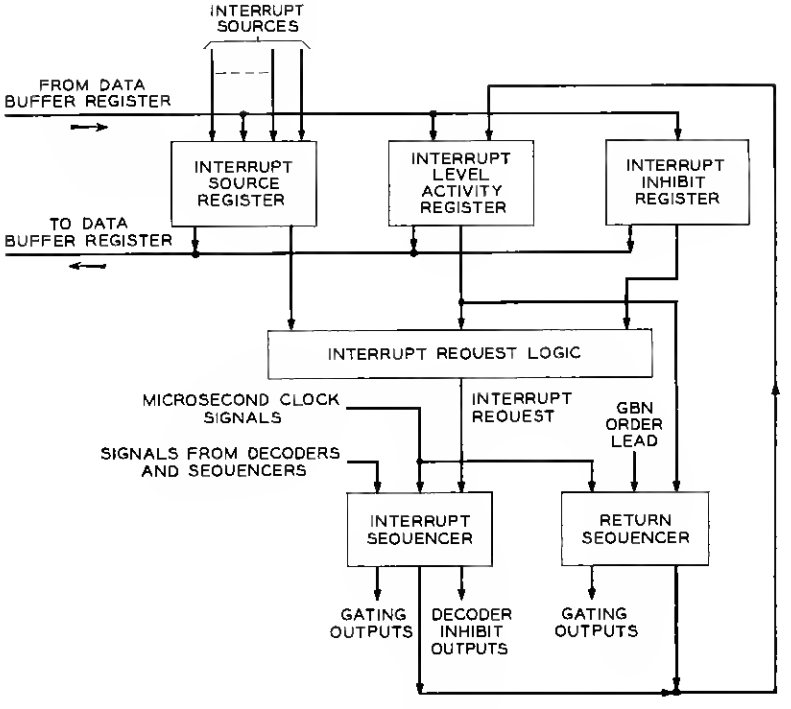


Fig. 28 — Interrupt system.

responding to each of the priority classes is a flip-flop within the interrupt-level activity register; whenever the interrupt system responds to an interrupt request, the wired transfer of program control is accompanied by the setting of the corresponding flip-flop in the interrupt-level activity register. The setting of flip-flops within the interrupt-level activity register also serves to inhibit the interrupt system from honoring interrupt requests from the level just served and all lower levels.

Assuming first that all flip-flops in the inhibit interrupt register of Fig. 28 are reset, the sequence of actions associated with each interrupt may now be described. When a given interrupt program sequence is required, a signal appearing on the corresponding interrupt source signal lead sets the corresponding interrupt source flip-flop. This signal propagates through the interrupt request logic and enables the interrupt request output conductor. The enabling of the interrupt request conductor is combined with clock signals and signals from the decoders and other sequencers to initiate the interrupt sequence. This assures that the interrupt allows any multicycle order or order following an

MCII or MKII instruction to go to completion before generating the wired transfer of program control. This consideration simplifies the hardware and program design for returning to interrupted program sequences.

Once activated, the interrupt sequencer carries out a number of functions extending over a period of several machine cycles; accordingly, the interrupt sequencer inhibits the order word decoder and buffer order word decoder outputs and generates independent gating signals to carry out a sequence of actions which include the following: (1) update the interrupt level activity register by setting the flip-flop in that register corresponding to the level interrupt currently being served (the interrupt request logic will then respond only to interrupt requests which may have a preassigned priority over the first interrupt program), (2) generate a transfer address corresponding to the entry point of the interrupt program sequence corresponding to the class of interrupt being served and gate this address to the program address register to effect this entry, (3) store the contents of the data buffer register in a first reserved location in the call store (this location depending upon the level of interrupt being served), and store the address of the instruction immediately following the last instruction executed prior to the interrupt.

Having completed these tasks, the interrupt sequencer returns to the inactive state, and the interrupt system is then responsive to further interrupt requests at higher levels. At this time, the entry to the corresponding interrupt program is made; this program begins the further storing of index registers, the logic register, etc., to complete the construction of the central control image in a set of reserved call store locations.

Upon completion of the required interrupt work functions, a program sequence restores the image of central control from the block of reserved call store locations. The interrupt program sequence then ends with a special return order (GBN), which activates another sequencer which completes the reconstruction of the central control image and transfers back to the interrupted program in an efficient three machine cycle sequence. This sequencer (also shown in Fig. 28) utilizes the interrupt-level activity register to complete the restoration of central control to the state occurring at the time of the interrupt. The sequencer must: (1) reinitialize the program address register to reenter the interrupted program at the proper point, (2) restore the data buffer register, and (3) reset the flip-flop in the interrupt-level activity register associated with the interrupt level from which the return is being made. Having completed these actions, the return sequencer advances to the

inactive state and is thereby made available for subsequent returns from other interrupt program sequences.

The inhibit interrupt register shown in Fig. 28, as its name implies, serves to selectively inhibit the response of the interrupt system to selected interrupt sources. The inhibit interrupt register is also a buffer bus register to which reading and writing access are provided. This register is used to selectively inhibit the interrupt sequencer response to interrupt signals during the execution of special test program sequences which, as part of their normal execution, cause the generation of interrupt source signals. The interrupt inhibit register also serves to inhibit interrupts due to repeated signals from defective subsystem units.

7.3 *Matching Circuits and Match Control Decoder*

In normal operation the duplicate central controls are executing identical orders within the same program sequences, and since the microsecond clock in both central control units is driven from one of two crystal oscillators,* the individual data processing steps for each order are closely synchronized in the two central control units. Normally, the two central controls are started by placing the same entry address in the program address register to simultaneously obtain and execute the same first program order. Each central control then continues in step with the other; the same data are read from memory or the scanners, the same data processing steps are performed on this data, and the outcome of each decision order is identical. Furthermore, certain trouble signals are cross connected between central control units so that any additional cycles inserted in one central control unit for remedial actions are accompanied by the insertion of the same number of cycles in the other unit.

The mode of operation just described is designated as the "in-step mode" and is utilized with the matching circuits to provide a continuing hardware check of the operation of the two central controls. This check consists of repeated comparisons of like information processing points in both central control units to obtain rapid detection of trouble conditions within either unit. The repeated comparisons are made with the matching circuits under the control of the match control decoder. A simplified block diagram of these matching circuits, decoders, and cross-connecting buses is shown in Fig. 29.

Within each central control are two internal match buses which pro-

* A flip-flop in each central control defines one unit as the active central control and the other unit as the standby central control. The crystal oscillator in the active central control drives the microsecond clocks in both units.

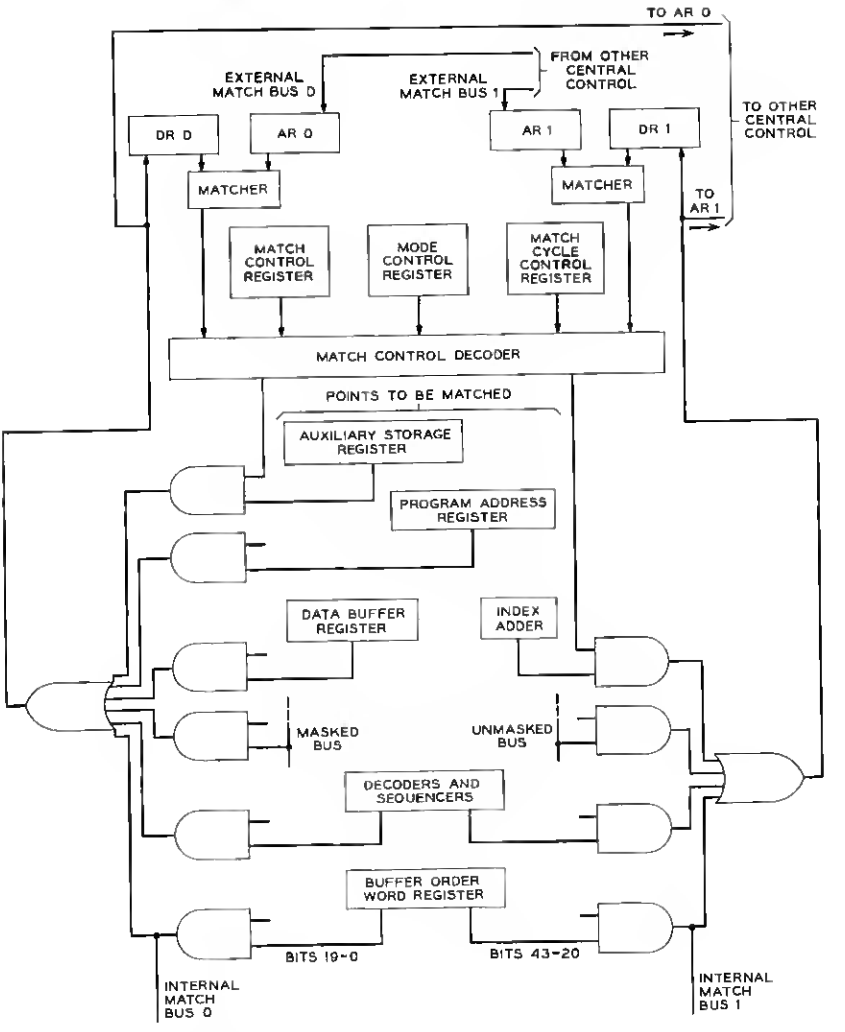


Fig. 29 — Central control matching circuits.

vide access to selected information processing points; these are labeled internal match bus 0 and internal match bus 1. Under control of the match control decoder, information from selected points is transmitted to these internal match buses, and from there other gates are enabled to place this information into internal match registers DR0 and DR1 and simultaneously transmit the sampled information via external match buses 0 and 1, respectively, to the other central control unit. The

match control decoders in both central controls are normally operating in step so that the information transmitted from the first central control to the second is stored in external match registers AR0 and AR1, respectively, in synchronization with the previously described gating actions. Two match circuits serve to compare the contents of AR0 with DR0 and AR1 with DR1; according to the state of the mode control register and the presence or absence of the match condition, the match control decoder generates the corresponding output signals. For example, when the matching circuits are employed in the routine matching mode, a selected sequence of common match points in each central control is matched at the rate of 2 matches per cycle; the detection of a mismatch condition generates a maintenance interrupt signal and further matching is automatically halted.

Since the matching circuits are limited to a maximum rate of two matches per machine cycle, the routine matching mode selects the specific sequences of internal points to be matched; the points to be matched depend upon the program and hardware actions being taken in central control to strategically examine those points most pertinent to the data processing steps that are occurring during a given machine cycle. Signals from the decoders and sequencers within central control are transmitted to the match cycle control register shown in Fig. 29. These signals set and reset specific flip-flops, which in turn direct the match control decoder and the selection of internal points for matching during each machine cycle.

The selection of points to be matched provides the detection of hardware troubles developing within central control as soon as the effect of that trouble would be communicated to other units in the switching system. It should be noted that the routine four-cycle match does not include the matching of all points to which the internal match buses have access. These additional points serve in other match modes described below.

7.4 *Maintenance Matching Modes*

A number of maintenance matching modes provide program-controlled access to the array of register buses and test points connected to internal match bus 0 and internal match bus 1. During the performance of certain maintenance programs, the routine matching mode is inhibited, and instead one of a number of maintenance matching modes may be selected to use the matching circuits to monitor test points within the standby central control or use the matching circuits and

connecting buses to communicate selected data from certain of these points from one central control unit to the other.

The maintenance matching mode to be executed and optional gating actions ensuing the detection of mismatch (or match) conditions are selected by the information placed in the mode control register. Certain of the maintenance matching modes match selected points at selected time intervals; information placed in the match control register determines these selections.

The selection of the routine matching mode or one of the maintenance matching modes is made by writing the selected word into the mode control register and the match control register shown in Fig. 29. To switch from one matching mode to another, a special flip-flop in both central controls is reset by a central pulse distributor command; this inhibits the response of the match control decoder to the matching mode currently specified. This step is followed by updating the mode control register and match control register to the new matching mode desired; following these actions, the CPD-controlled flip-flop is again set and the match control decoder is responsive to the new mode. A description of each of these modes and its use appears in a companion paper.[14]

7.5 *Emergency-Action Sequencer*

The preceding sections describe a number of hardware checks and allude to both hardware and program remedial actions in response to circuit troubles detected in the central processor. These remedial actions include program sequences which are calculated to isolate circuit troubles to a particular unit and to control any required switching of units to obtain a working system. The execution of these sequences is in itself possible only if the active central processor includes an operable combination of the central control, certain program stores and the interconnecting bus system. That is, if the fault itself lies in one of these units, the central processor may be incapable of performing the necessary rearrangements. The emergency-action sequencer serves as a hardware checking and corrective means to handle this problem.

A block diagram of the interconnections between the emergency-action sequencer and its inputs and outputs is shown in Fig. 30. The inputs consist of four hardware checks; the failure of any one or more of these checks generates trouble signals to activate the emergency-action sequencer. These signals are de-connected to the inputs of a monostable pulse circuit; the output of this monostable pulse circuit is connected to a series of monostable pulse amplifiers to provide a sequence of output

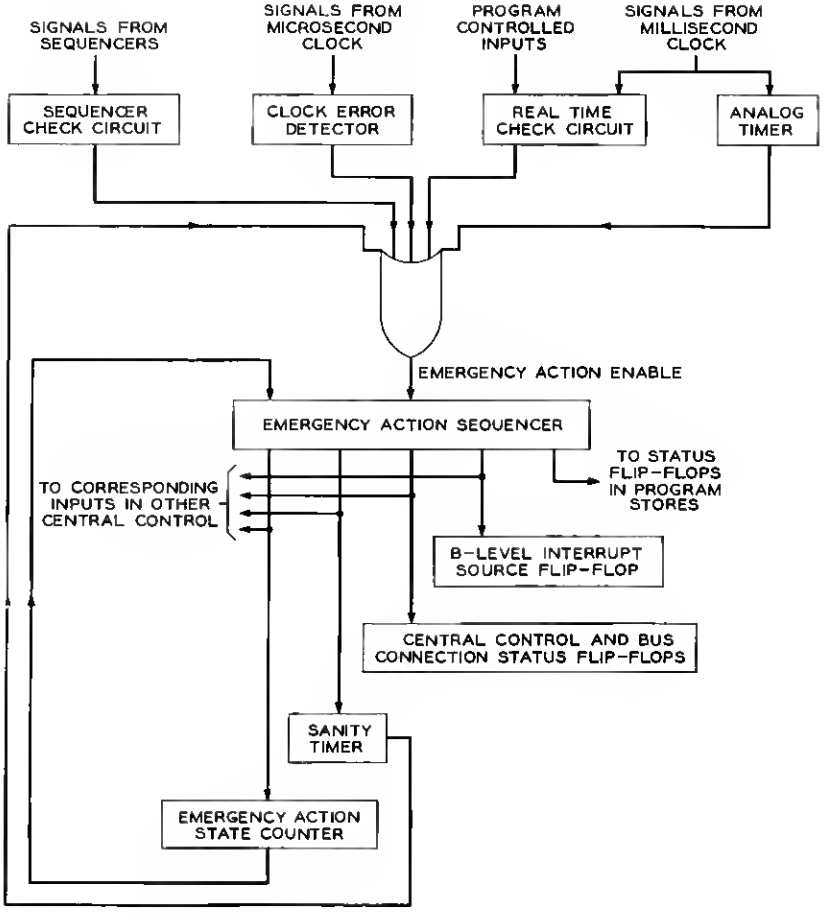


Fig. 30 — Central control emergency-action system.

pulses. The emergency-action sequencer is therefore operable in the presence of circuit troubles in the microsecond clock.

Each sequence of output pulses generated by the pulser and delay line defines an emergency action. Each emergency action increments an emergency-action state counter, initiates a B-level interrupt, and sets and resets selected status flip-flops in the duplicate central control. Switching these flip-flops results in a corresponding rearrangement of active and standby central controls, program store input and output connections, and selection of connecting buses. The rearrangements made during each emergency action depend on the internal state of the

emergency-action state counter. It is quite possible that a given rearrangement of duplicated units in the central processor will not result in an operable combination of subsystems; the result will be the reactivation of the emergency-action sequencer which, under control of the emergency-action state counter, forms a new arrangement.

When an operable combination of units have been connected together to become the active central processor, the execution of a test program (initiated with a B-level interrupt) can be completed in approximately 100 machine cycles. If the active central processor contains faulty equipment which inhibits the proper execution of the test program sequences, the sanity timer will recycle in approximately 128 machine cycles. Such recycling reactivates the emergency-action sequencer.

A more detailed description of sequencer actions and the additional hardware and program checks performed with the emergency-action sequencer are covered in a companion paper.[14]

7.6 *Maintenance Orders*

Included in the order structure of the central processor are classes of orders designed specifically for use in maintenance program sequences to obtain test results or to place test signals within central controls, call stores, and program stores. These orders perform special gating actions that are either inconvenient or impossible to obtain with combinations of other orders described above. The maintenance orders include the following classes: (1) G- and H-mode memory reading and writing orders, (2) control mode memory reading and writing orders, and (3) miscellaneous test and test signal orders.

7.7 *G- and H-Mode Memory Orders*

As previously noted, all program and data information is duplicated; each word stored in one program store unit or call store unit is also stored in another store unit. There are many situations where initial installation requirements or growth in an office require an amount of duplicated semipermanent and/or temporary memory that could be satisfied with an odd number of store units. The duplication scheme employed in the No. 1 ESS central processor permits the use of an odd number of units to gain economy over a system using only an even number of stores.[14] In this duplication scheme each store unit is divided into two blocks of memory. Each block or half-store is assigned a different code. One block in the memory unit is designated the H block; the remaining block is labeled the G block of memory. All the information appearing in the H block of one store unit is duplicated in the G block of another store unit.

Commands to read or write in memory include a code and address. Each code and address corresponds to one word of information in memory, but since each word is duplicated in the memory, the code and address correspond to two memory locations, one appearing in the G block of one store unit and the other location appearing in the H block of a different store unit.

The usual communication of data and program words between the central control and connecting stores is accomplished with normal mode commands. Each normal mode command is capable of generating a simultaneous response in the store units containing the G and H duplicate memory locations; however, no store unit simultaneously receives two commands to which it is to respond, and no central control unit simultaneously receives both of the duplicate responses.

When troubles are detected in the communication of information between the stores and the central controls, the remedial actions include program sequences to examine individual store units; it is desirable to read or write test information specifying the store unit which is to respond to these commands. Consequently, the order structure includes G and H memory reading and writing order words or instructions that specify which duplicate location is to be read or written. These memory reading and writing orders are processed in central control like the normal mode reading and writing orders: the codes and addresses are generated in the same fashion, but the command includes G- or H-mode signals. For example, a G-mode memory reading order will obtain information from only the duplicate unit which contains the G image of the memory location specified by the code and address. H-mode memory reading and writing commands make a similar distinction.

The G- and H-mode memory reading orders include additional features when applied to the reading of data in program stores. Certain of these orders call for the reading of data from the G or H locations, respectively; when the data is obtained from a program store, the readings are accepted and processed as valid data without carrying out any rereading or correcting steps as indicated by the check circuit. Other G and H memory reading commands, when applied to program store memory locations, permit the correction of data as required, but no rereadings may take place. The G and H memory reading and writing instructions just described are given in Table II.

7.8 *Control Mode Orders*

Control orders resemble the normal mode memory reading and writing orders in that central control carries out the reading or writing of data in

TABLE II — G AND H MEMORY ORDERS

Mnemonic Representation	Order	Available Options
HKU	Read H image of specified memory location; no remedial actions for invalid responses; data reading replaces contents of accumulator	Indexing; index register modification options, including incrementing, W, and register S options; product (PL and PS) masking and complementing of data reading
GKC	Read G image of specified memory location; remedial action limited to single-error correction of program store data readings; data reading replaces contents of accumulator	
XGKU	Read G image of specified memory location; no remedial actions for invalid responses; EXCLUSIVE-OR of data reading and accumulator contents placed in accumulator	
XHKC	Read H image of specified memory location; remedial action limited to single-error correction of program store data readings; EXCLUSIVE-OR of data reading and accumulator contents placed in accumulator	
BG, BH	Place contents of data buffer register in specified G (H) call store memory location; no remedial actions for invalid responses	Indexing; index register modifications listed above
FG, FH KG, KH, LG, LH	Place contents of index register F (accumulator K, logic register L) in specified G (H) call store memory location; no remedial actions for invalid responses	Indexing; index register modification options listed above; product (PL and PS) masking; complementing and insertion (EL and ES) masking of data to be stored

such units as the program store and the call store. These orders differ from normal mode memory order in that: (1) when commands are transmitted, the mode signals which appear as part of these commands indicate the control mode, and (2) control mode memory writing orders may be used also to treat flip-flop registers in the stand-by central control as memory locations in a call store.

Control orders are designed to provide the convenient setting, resetting, and reading of status flip-flops and other test points in the program stores, call stores, and standby central control.

The control mode orders are listed in Table III; this table includes comments regarding the specific application of certain control mode

TABLE III — CONTROL MODE ORDERS

Mnemonic Representation	Order	Available Options
NB	Read specified control location; place reading in data buffer register	Indexing; index register modification options, including incrementing, W, and register S options
NF, NJ, NK, NL, NX, NY, NZ	Read specified control location; place reading in index register F (J, K, L, X, Y, Z)	Indexing and index register modification options: product (PL and PS) masking and complementing of data
NBTA, NBTB	Special control reading orders to test call store address translators; translated address placed in data buffer register	Indexing and index register modification options
BN	Place contents of data buffer register in specified control location	
FN, JN, KN, LN, XN, YN, ZN	Place contents of index register F (J, K, L, X, Y, Z) in specified control location	Indexing and index register modification options: product (PL and PS) masking, complementing, and insertion (EL and ES) masking of data to be stored
WNPS	Transmit control command (specifying) location and data for control flip-flops in a specified program store	Indexing

orders. For example, the order WNPS is designed specifically for writing information into duplication status and test flip-flops within the program stores. The program stores are the semipermanent memory of the system. No on-line writing of information within the twistor memory is possible and therefore none of the previously described writing orders has access to the program store. The WNPS order is executed by sending a command on the program store bus which indicates a control mode writing operation, and part of the address transmitted is treated by the responding program store as the data to be placed in control flip-flop registers within that store.

7.9 *Miscellaneous Maintenance Instructions*

In addition to the two classes of special memory reading and writing instructions described above, there are a number of specific orders, including some normal mode memory reading and writing orders designed

specifically to create trouble conditions not encountered in other normal memory commands or to initiate special tests.

Executing the order WV causes a data word to be transmitted to a special V register in central control; the outputs of this register are then transmitted as half-microsecond pulses to special points within the central control and via cable drivers and connecting twisted-pair cables to the other central control unit. This order is used, for example, for transmitting signals from the active central control to standby central control to:

(1) start or stop data processing in the standby central control;

(2) reset certain registers in the standby central control unit such as the buffer order word register and the order word register, or

(3) generate maintenance interrupt signals in the standby central control.

A mismatch sampling order, EMMS, concurrently carries out a number of information processing steps to initiate the mismatch sampling mode as described in a companion paper.[14]

The remaining miscellaneous maintenance orders (BMAP, BMOP, MBOP, and MBCS) comprise normal memory reading or writing orders which are specifically designed for exercising or examining the parity generation and check circuits in both central control and the call store.

VIII. TIMING CONSIDERATIONS

The central processor is a synchronous data processing system; a microsecond clock in central control provides clock pulses defining a machine cycle and intervals or phases within that cycle. A significant aspect of both the logical organization and detailed circuit specifications of the central processor is the integration of the response times of program store and call store systems and the multiphase data processing steps of central control in response to each of a diversity of program orders. This integration began with the preliminary design of a central control and its order structure.

The classes of orders considered for inclusion in the order structure consist primarily of different combinations of meaningful data processing operations or steps such as indexing, index register modification, and the placing of a data word obtained from memory into a specific index register in central control. Each major data processing step is assigned to one or more clock phases; the minimum time for a clock phase is fixed according to the maximum propagation time of information through the longest logic chains corresponding to the data processing

steps assigned to that phase. These phases are then fitted into a machine cycle which approximately equals the minimum cycle times of communication between the central control and both the call store and program store; the relative placement of each of the phases is dictated by the time of appearance of related data processed in central control to form commands, addresses and data for communications between central control and its stores.

In the design of classes of order words and the determination of the number of clock phases, an extra clock phase is provided by overlapping operational steps of successive orders. To maximize the average data processing rate, most of the orders are designed to be executed at the rate of one order per machine cycle. Certain orders, such as transfer orders and orders to read data words from the program store, require more time and are designed to fit into a timing framework of two or more machine cycles.

Detailed timing studies based on min-max component tolerances served to provide a "paper simulation" of the central processor. This simulation revealed that a number of logic chains and communication paths are limiting in fitting all the operations steps into a 5.5-microsecond clock cycle; accordingly, an improvement of any one logic chain or subsystem could not materially increase the central processor's data processing speed capabilities.

8.1 *Single-Cycle Call Store Memory Orders*

The central processor is designed to execute most of its sequences of program orders at the rate of one order per machine cycle; such orders are referred to as "single-cycle" orders. Multicycle orders require additional time (2 to 4 machine cycles, depending on the order) and are executed with the aid of special sequential control circuits described later in this article. The data processing time of the single-cycle orders determines the machine cycle time and the allocation of phase intervals within this cycle; included in this class are memory reading and writing orders which receive and transmit data from memory locations in the call stores.

Call store memory reading and writing orders comprise an estimated 60 to 70 per cent of the instructions executed in call processing program sequences; the ability to execute these orders at the rate of one order per machine cycle defines a machine cycle, which may be limited by one or more of three considerations:

(1) the maximum repetition rate of obtaining program order words from a program store system including up to 6 program stores;

(2) the maximum repetition rate of a call store system of up to 37 call stores in response to a random sequence of reading and writing commands generated in the execution of a sequence of call store memory orders; and

(3) the maximum turn-around time at which a data reading can be obtained from a call store and combined in an index adder to generate an address for use in a call store reading or writing command.

An additional requirement in allocating clock phase intervals is noted in fitting call store writing orders into the machine cycle. An upper bound is placed on the interval from the beginning of the clock phase which initiates indexing (to obtain a call store address) to the beginning of the clock phase which moves the data to be stored from an index register to the data buffer register.

There is no difficulty in fitting the data processing steps of the non-memory data processing orders into this framework; the only orders employed in significant numbers in the program sequences requiring more than one machine cycle for their execution are orders to transfer program control and memory reading orders which obtain data words from the program store.

8.2 *Phases of the Machine Cycle*

The overlapped execution of sequential program orders combined with 3 basic data processing phases per machine cycle provides four data processing intervals per instruction. This is a sufficient number of intervals for all the data processing steps required for transmitting information via one or both of the unmasked bus and masked bus from one register to another through logic combining circuits such as the index adder or the mask and complement circuit.

Fig. 31 shows how call store memory orders are fitted into the overlapped execution of single-cycle orders. An order is obtained from a program store during phase 1 of cycle 1 in response to a program store command transmitted from the central control in the preceding machine cycle. At the beginning of this phase, the buffer order word register is reset to erase the preceding order word, and the register inputs are connected to the program store response bus. Depending on the response time of the program store unit and its distance from central control, the order word arrives at some time during phase 1. In this interval, the command and address for the first succeeding program order are transmitted to the program store.

Phase 2 and 3 clock pulses are applied to the buffer word decoder to control indexing and index register modification during cycle 1. The indexing addition is completed towards the end of phase 2 and the sum

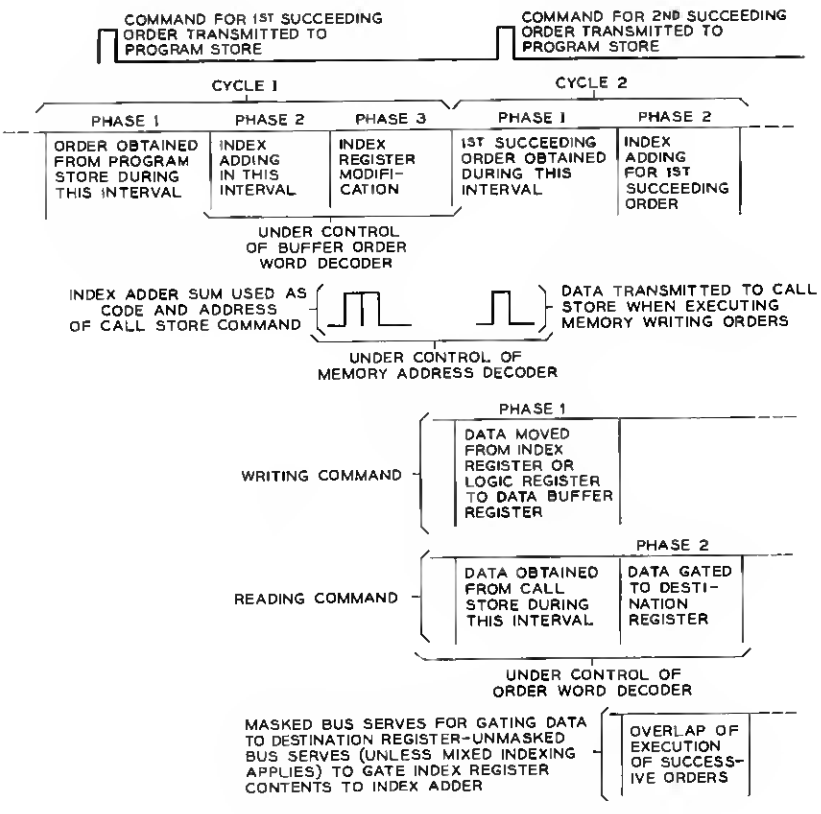


Fig. 31 — Timing of execution of single-cycle memory orders.

is gated to an index adder output register (contained in the index adder). This register serves as an address register for call store memory orders. Index register modification is performed during phase 3, and the call store command and address are concurrently transmitted under control of the memory address decoder to the call store.

At the beginning of phase 3 of cycle 1 the call store memory order depicted in Fig. 31 is transmitted from the buffer order word register to the order word register. Phase 1 and phase 2 clock pulses applied to the order word decoder and memory address decoder carry out the remaining steps for the order. Two sequences of these remaining steps are shown: the first sequence is that of a call store memory writing order, and the second corresponds to a call store memory reading order.

Call store memory writing orders select the contents of one of the

seven index registers or the logic register to be transmitted as data to the eall store. The data buffer register in eentral control is conneeted to the eall store data transmission and call store response buses; aeeordingly, the contents of the selected register are transmitted via the unmasked bus, the mask and complement circuit, the masked bus and the insertion mask to the data huffer register. A parity generator eonneeted to the outputs of the data buffer register produces a parity bit whieh is then transmitted simultaneously with the 23-bit data word to the call store to complete the call store memory writing order.

Call store memory reading orders employ phase 1 of cycle 2 to accept data words hy connecting the inputs of the data buffer register to the call store response hus. The data buffer register is reset at the beginning of phase 1 and the data word is reeeived at some time during phase 1, depending on the response time of the call store unit and the distance between that unit and the central control.

During phase 2 of cycle 2, the data reading is transmitted to the destination register selected by the memory reading order by gating the reading through the data huffer register and the mask and complement circuit onto the masked bus.

Concurrently with the eompletion of the memory reading order, the first suceeeding order begins its indexing. That is, phase 2 of eyele 2 includes the transmission of the contents of a selected index register to the index adder via the unmasked bus. Certain pairs of orders occur where the first order reads data from a call store and the seeond order is a call store memory order which selects as its index register the destination register of the first order. (These pairs of orders require the previously described mixed indexing.) To execute such pairs correctly, the lower bound on the machine cycle time must equal or exceed the maximum round-trip time of call from call store command to response, ineluding the indexing addition in central control.

Indexing and index register modification of all other orders are also performed during phase 2 and phase 3 of the first cycle, as shown for memory orders in Fig. 31. W class combining orders move the data word to a specified destination register during phase 1 of the second cycle; shift and rotate orders are also completed in the same interval. The earlier completion of these orders (as compared to memory reading orders) permits the accumulator to be seleeted as the index register in the succeeding order; such a selection is not feasible when the first order is a memory reading order moving data to the aeeumulator, since the accumulator will not contain the correct result in time for indexing during phase 2.

The data processing steps of orders such as UWX, PMZ include moving the contents of an index register (X,Z) to the logic register; this step is performed using the unmasked bus during phase 3 of the first cycle. The unmasked bus is available, since the index register modification step requires the use of only the masked bus; accordingly, the two steps may occur concurrently, as required by the order.

Whenever an order specifies that the data word is to be transmitted to the logic register (PS or ES masking) this step is accomplished during phase 3 of the first cycle under control of the buffer order word decoder.

Decisions are made by the order word decoder for regular transfer orders at the beginning of the second cycle; decisions for early transfer orders are made at the beginning of phase 3 of the first cycle. In both cases, the decision to transfer is implemented by activating a sequencer at the stated times.

8.3 *Basis of Timing Specifications*

Figs. 32 and 33 show the response times of the program stores and call stores in terms of minimum and maximum calculated response times. To provide sufficient spacing for large numbers of program stores and call stores, bus lengths from a few feet to a maximum of one hundred feet were assumed for each bus. The buses are all twisted-pair cables having a delay constant of approximately 2 nanoseconds per foot.[12] A single program store command bus system connects both central control units to all program stores; each store has a lightly coupled set of pulse receivers connected in series with the bus system, and the bus ends with a terminating resistor. These receivers add a small fixed delay to the bus system, 1.5 nanoseconds per receiver, which is insignificant in most timing considerations. Similarly, a single program store bus system connects all program store memory outputs (cable drivers are shunt connected) to inputs in both central controls. A similar arrangement of shunt and series connections of cable drivers and pulse receivers connects the central controls and the call stores. These connections include a call store command bus system, a call store response bus system, and a call store data transmission bus system.

To predict the delays of logic chains in central control, the characteristic minimum and maximum delay times of the low-level logic (LLL) AND-NOT gate[6] and the medium-current logic (MCL) AND-NOT gate were determined, as tabulated in Fig. 34. These limits include allowance for lead capacitance, and wiring rules restrict the actual capacitances to less than this allowance.[6] The characteristics of the cable driver (CD), cable pulse receiver (CPR) and clock pulse output amplifier (CPOA)

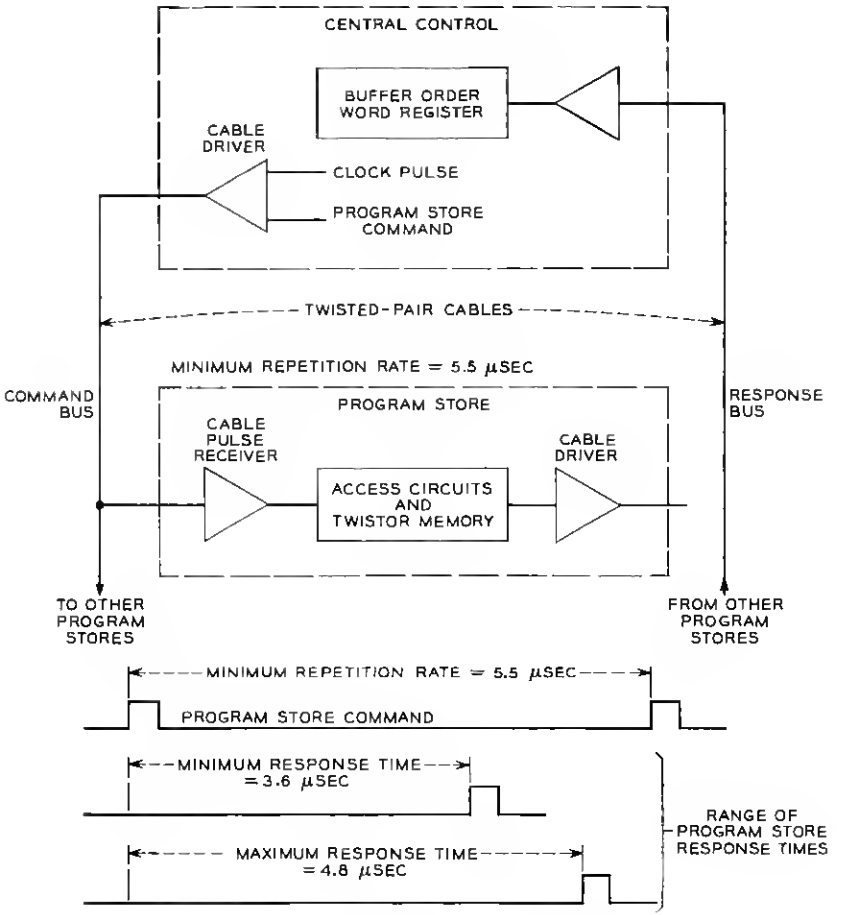


Fig. 32 — Program store timing tolerance limits.

gates complete the information needed for calculating propagation delays of central control.

In certain timing chains, minimum pulse widths become a part of the design considerations, and therefore a pulse-shortening characteristic for each class of gate is included in Fig. 34. It is assumed that the turn-on time of an LLL gate or an MCL gate cannot exceed its turn-off time; a consequence is that reduction in the width of a pulse propagating through a number of stages of logic gates due to differentials in turn-on and turn-off times is calculated to occur only in alternate stages which have negative-going pulse inputs.

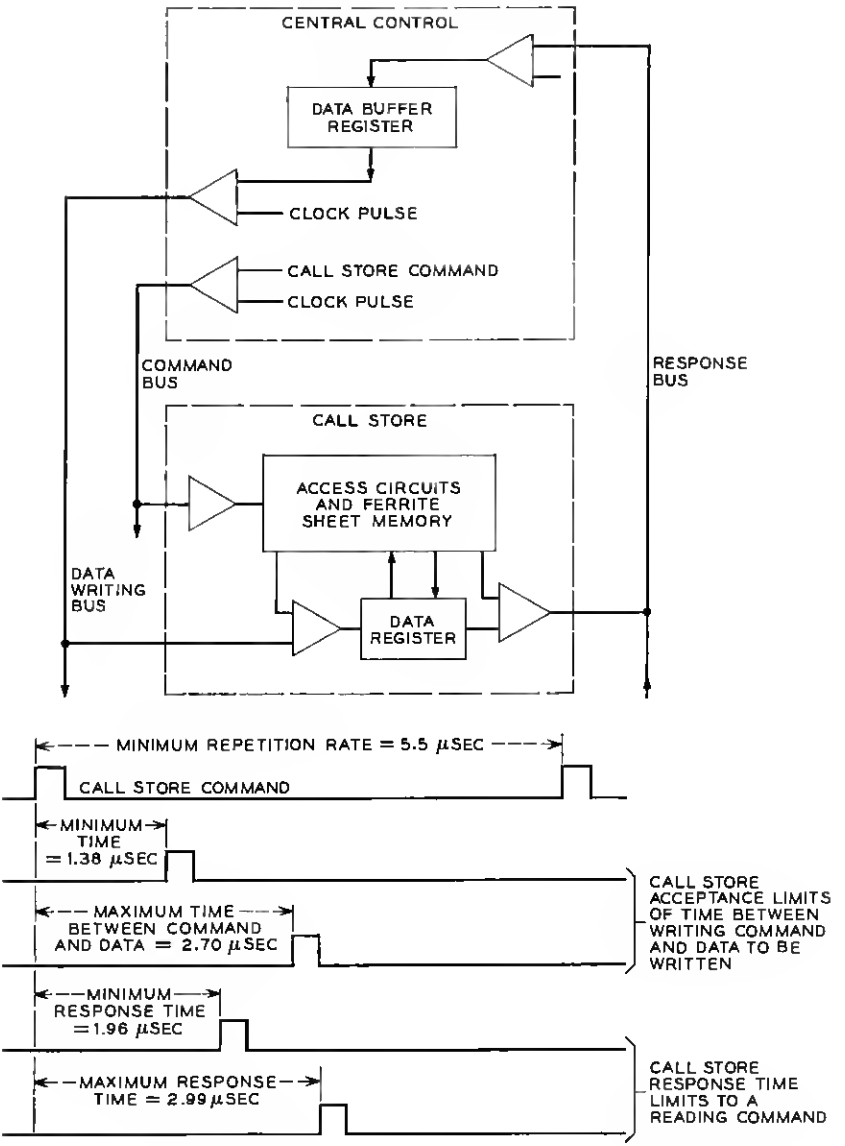


Fig. 33 — Call store timing tolerance limits.

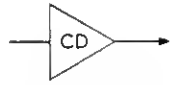
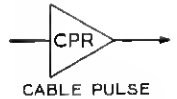
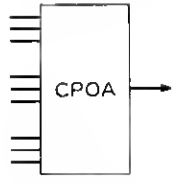
	TURN-ON TIME IN NANOSECONDS	TURN-OFF TIME IN NANOSECONDS	MAXIMUM SHRINKAGE OF PULSE WIDTH
	10-65 NS	10-65 NS	55 NS - FOR NEGATIVE-GOING INPUT PULSES ONLY
	10-75 NS	20-85 NS	75 NS - FOR NEGATIVE-GOING INPUT PULSES ONLY
	10-75 NS	10 NS - UNSPECIFIED	NONE
	10-65 NS	10 NS - UNSPECIFIED	NONE
	10-75 NS	10-75 NS	20 NS

Fig. 34 — Logic package timing characteristics.

8.4 *Microsecond Clock Characteristics*

The multiphase clock in central control defines the machine cycle of 5.5 microseconds and the three phases within the machine cycle. In addition, a number of shorter pulses are provided by the clock to carry out the gating of commands and addresses from central control through cable drivers onto twisted-pair cables connecting to stores, central pulse distributors and peripheral units such as scanner and network controllers. Other short pulses are required for resetting registers and gating informa-

tion from one flip-flop register to another at times other than the three principal phases; data processing results are thereby transmitted as they are completed. Half-microsecond pulses spaced at quarter-microsecond intervals over the entire machine cycle are provided to permit maximum flexibility in the logic design; timing chains were easily modified as additions and alterations were made in the development of the logical organization of the central processor. This large array of pulses thus satisfies the need for (1) selecting optimum gating times for internal data processing transmission and (2) obtaining a useful minimum pulse width (where reasonable clock pulse tolerances are a consideration) for (a) gating information from one register to another, (b) resetting a register, and (c) transmitting a pulse of sufficient width over twisted-pair bus systems to the most distant stores, central pulse distributors, and peripheral units.

The microsecond clock for central control includes a crystal-controlled 2-megacycle oscillator driving an 11-stage counter. The outputs of the counter drive logic chains which translate the states of the counter into the array of half-microsecond pulses and the basic phase pulses are shown in Fig. 35. The machine cycle is divided into 22 quarter-microsecond

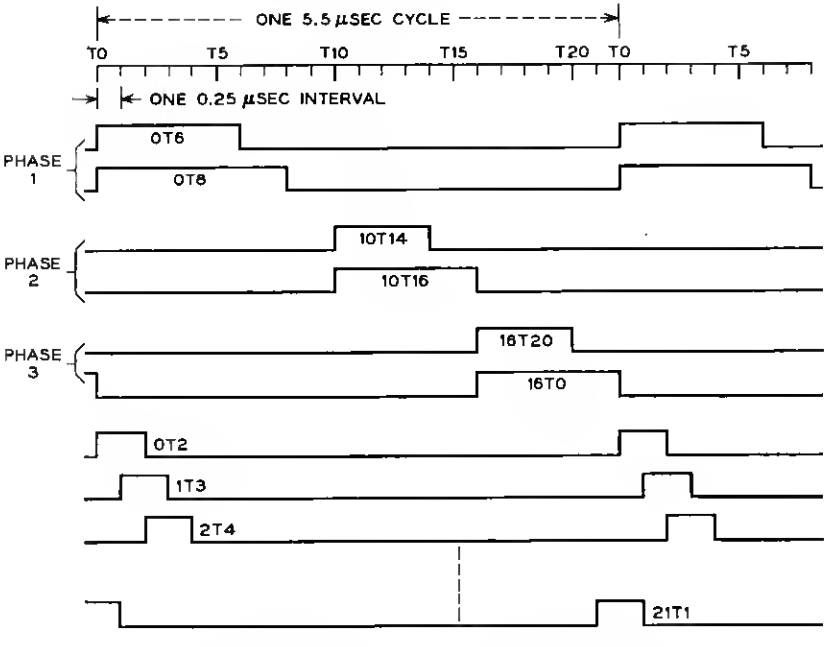


Fig. 35 — Phases of central control microsecond clock.

intervals and the beginning of each interval is denoted as T0, T1, · · · , T21. Each clock pulse begins and ends at one of the times T0, T1, · · · , T21, so that each clock pulse is labeled ATB, where A is a number corresponding to the time of the leading edge of the pulse and B corresponds to the trailing edge time.

Both central controls contain a complete clock, but the oscillator in the active central control drives the 11-stage counter in both central control units to keep the units closely synchronized. To keep the counters in step, a phasing signal is transmitted from the active central control counter to the standby unit once every machine cycle.

The tolerances on the clock outputs in each central control and the cross-connection tolerances are tabulated in Fig. 36. Fig. 35 depicts an ideal set of clock pulses and the table indicates additional delays from that ideal. That is, the figure represents all minimum delay conditions in the clocks, and only positive tolerances appear in Fig. 36. This approach simplified the many calculations to be made in the design. Only minimum/maximum values were substituted into delay equations, rather than nominal values plus or minus a tolerance figure. Examples of this technique follow.

8.5 *Sample Timing Calculations*

The half-microsecond pulses are employed in transmitting multibit commands and addresses to units connecting to the central control. To meet high fan-out requirements, a clock output amplifier is usually interposed between the clock pulse output and the array of cable drivers to be pulsed. This connection is exemplified in Fig. 37 using the pulse 0T2 for purposes of illustration.

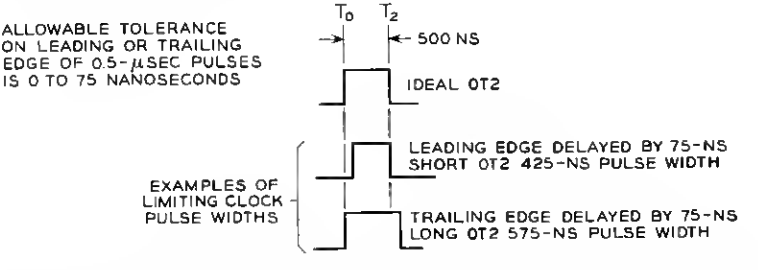


Fig. 36 — Microsecond clock tolerance limits. Allowable tolerance on leading or trailing edges of bus phase pulses (0T6, 0T8, 10T14, 10T16, 10T20, 10T22) is 0 to 150 nanoseconds. Allowable tolerance of propagation of oscillator signal from active to standby central control is such that idealized phases of Fig. 35 in standby central control lag by 0 to 75 nanoseconds.

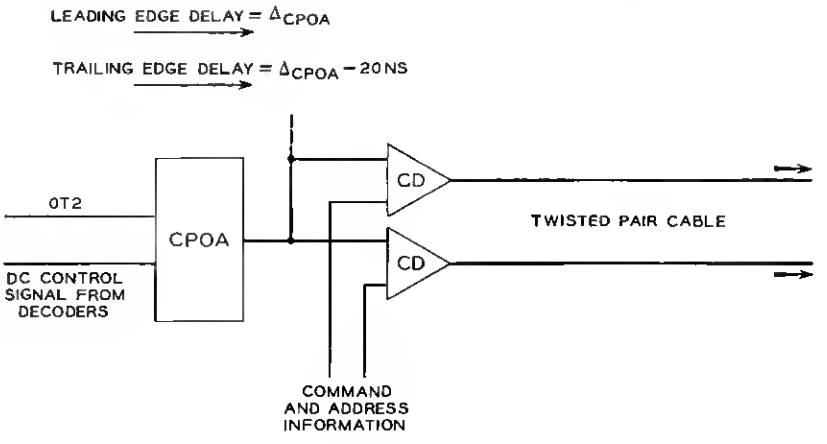


Fig. 37 — Clock pulse transmission chains.

A timing calculation was made to determine if such an arrangement would always provide command and address pulses of sufficient width (250 nanoseconds minimum) to a receiving unit located at the maximum allowable distance from central control. To determine this minimum pulse width, one first assumes the shortest allowable clock pulse and then calculates the maximum pulse shortening in logic chains and cables.

Referring to Fig. 36, the latest leading edge of the pulse would occur at T0 + 75 nanoseconds, and the earliest trailing edge would appear at T2, yielding a minimum clock pulse width of 425 nanoseconds. According to Fig. 34, the clock pulse output amplifier may shorten this pulse as much as 20 nanoseconds, and the cable driver would contribute no pulse shortening. Thus the minimum pulse width at the cable driver output is 405 nanoseconds.

Empirical equations for the shrinkage of the width of current pulses on a twisted-pair cable with an arbitrary number of series-connected cable receiver transformers as a function of that number (N) and the length of the twisted-pair cable (L in feet) have been derived.[16]

$$\text{Shrinkage in nanoseconds} = 50 \ln \left(\frac{950}{950 - L - 5N} \right). \qquad (1)$$

The bus system connecting central control to the peripheral units may be required to serve a large number (N) of such units and the bus length be correspondingly large. With limits of $N = 50$ and $L = 450$ feet the maximum shrinkage would be approximately 70 nanoseconds. Thus the

minimum pulse width at the input of the most distant peripheral unit would equal or exceed (405 − 70) = 335 nanoseconds, which meets the peripheral unit minimum pulse requirements.

The previously described round trip of one call store reading to be used in generating a call store command is outlined in Fig. 38 and results in a round-trip time of 5.36 microseconds, which fits into a 5.5-microsecond cycle. The calculated round-trip time does not allow for a slight delay in the loop deriving from clock pulse gating between the index adder and the index adder output register. Further, it should be noted that the late clock pulse arriving at point Y cannot be considered, since the memory reading may be obtained in response to a call store command generated in the active central control and utilized in the standby central control for generating a call store address. The 10 LLL delays in the index adder represent the maximum delay of carry propagation signals. Other adder designs with shorter delays were considered; however, smaller delays

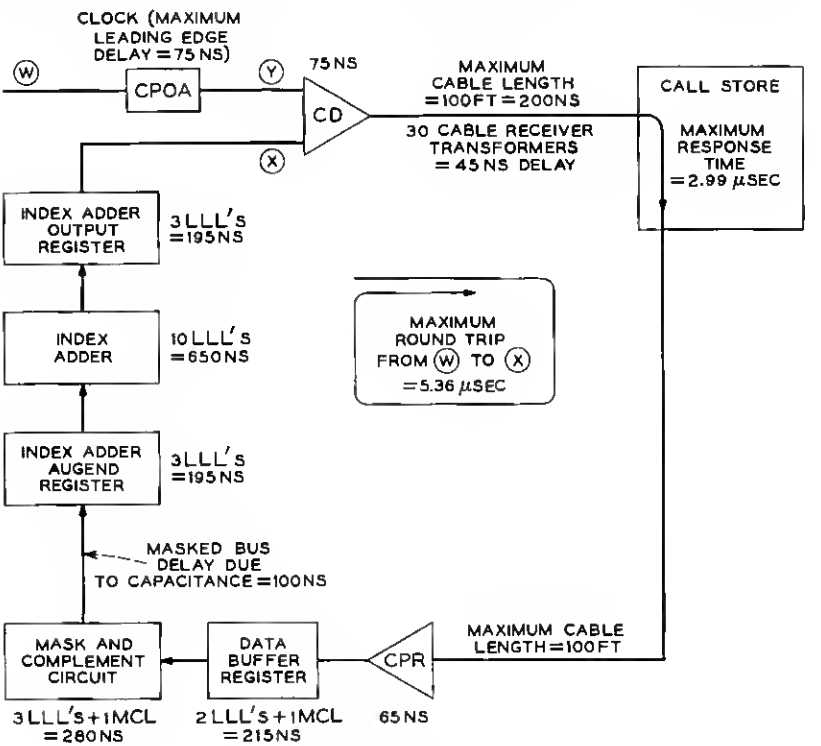


Fig. 38 — Call store round-trip time.

and a slightly shorter machine cycle could be achieved only with much larger adding circuits.

Other timing chains not described here also work with the 5.5-microsecond cycle but could be made to work with a shorter cycle only with additional logic circuits. Other limiting cases not described in detail here include:

(1) communicating commands to peripheral units once every two machine cycles to establish a maximum rate for supervisory scanning of customer lines,

(2) round-trip times of call store memory reading orders including the cross connection of associated check failure signals between central control units, and

(3) various combinations of related sequences of data processing steps in the execution of consecutive orders.

Fig. 39 illustrates some of the constraints that govern the subdivision of the clock phases within the machine cycle. The longer of the pair of pulses for each phase must not overlap, but time intervals between phases is allowed.

Phase two data processing includes the last step of a memory reading order where the contents of the data buffer register are transmitted via the mask and complement circuit and the masked bus to a selected destination register. This step requires a one-microsecond interval to complete the required propagation of information, and therefore phase two comprises the one-microsecond bus sampling interval 10T14 and a corresponding source-to-bus interval 10T16. A similar propagation delay requirement for index register modification options applies to phase

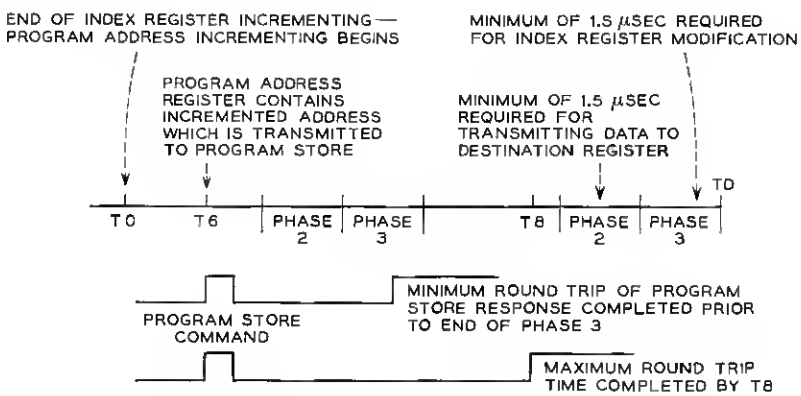


Fig. 39 — Fitting of clock phases into a machine cycle.

three, where the contents of the index adder output register are transmitted via the mask and complement circuit and the masked bus to the selected index register. Accordingly, clock pulses 16T20 and 16T22 define phase three.

Phase two and phase three require 3.0 microseconds of the machine cycle, and the incrementing circuit is employed throughout this interval for index register incrementing. The end of this interval is represented as T22 or T0, the beginning of the next machine cycle. Starting at this time, the contents of the program address register are transmitted to the incrementing circuit; the incremented quantity is then returned to the program address register and transmitted as part of a program store command to obtain the next program order word in sequence. The maximum time for the above steps is 1.5 microseconds, so that the program command is transmitted during 6T8 to the program store command bus. The maximum round trip returns the program word by T8 of the following cycle. This leaves a reasonable margin of 500 nanoseconds for signals to propagate through the buffer order word decoder before beginning gating actions starting at T10.

Calculations indicate that the minimum round-trip time to the program store returns the next program word before the buffer order word decoder has completed gating actions on the immediately preceding order. To circumvent a conflict on the use of the buffer order word register, an auxiliary register is placed between the program store response bus and the buffer order word register. For phase three gating actions, the operation field of the order in the buffer order word register must be retained, but not the accompanying 21-bit data field portions and 7-bit Hamming and parity portions of that program word. Accordingly, only the 28 corresponding cells of the buffer order word register are reset and connected to the program store response bus system during phase three; the auxiliary register serves to receive and retain the 16-bit operation field of the succeeding order until the buffer word decoder has completed phase three data processing steps for the current order. The contents of the auxiliary register are then transmitted to the buffer order word register during phase one of the following cycle in preparation for data processing steps beginning at phase two.

The above sample calculations are representative of those made as part of determining the realizability of the logic organization and detailed circuit design of the central processor for No. 1 ESS. These calculations served not only as a check of feasibility of proposed circuit design but also as guides in determining specifications for:

(a) tolerance limits on duplicate multiphase clocks driven from one of two crystal-controlled oscillators,

(b) maximum lengths of twisted-pair cable connecting central control to its communicating units, and

(c) tolerances between individual pairs and between buses of twisted-pair cables.

IX. DESCRIPTION OF SEQUENCER FUNCTIONS AND DESIGN

As indicated in Fig. 31, the buffer order word decoder, mixed decoder, and order word decoder serve to execute sequences of single-cycle overlap orders at the rate of one order per 5.5-microsecond machine cycle. The overlap of data processing occupies only one phase of the machine cycle, but when the generation of commands to obtain program order words is considered an additional degree of overlap is evident. The execution of certain orders (as well as hardware remedial actions such as the automatic rereading of information from the program stores and call stores) requires either the insertion of additional cycles of data processing or an extension of overlap to provide the necessary time. In providing these additional gating actions in central control a number of sequencers are provided; each sequencer carries out a specific class of data processing or remedial actions. Those sequencers which insert extra machine cycles inhibit the decoders to momentarily halt the flow of instructions obtained from the program store, and in this interval additional gating actions are carried out. Other sequencers do not insert cycles but extend the overlap by operating concurrently with the decoders; the decoders continue processing sequences of orders without regard to sequencer gating actions; in this latter instance, programming restrictions are applied to prevent conflicts in the overlapping flow of data processing. A brief description of the functions associated with some typical sequencers follows.

9.1 *Data Reading Sequencer*

All of the memory reading orders previously described (including G- and H-mode memory reading orders and control mode memory reading orders) may obtain data not only from the call stores but from any location within one of the program stores. Each of these memory reading orders generates a code and address during its indexing cycle, and the code so generated determines the memory location to be read and hence whether the data word is to be obtained from a call store or a program store. If the code and address refer to a call store location, then the central control carries out the single-cycle memory reading operation previously described. Whenever the code address refers to a program store

location, the data reading sequencer is enabled to obtain these data. Once this has been accomplished the data reading is in the same position as data obtained from the call store (i.e., placed in the data buffer register), and the data reading sequencer returns to the inactive state. The order word decoder then proceeds to complete the moving of the data from the data buffer register to the destination register specified in the memory reading order.

As the data reading obtained is returning from the program store the data reading sequencer simultaneously returns the address of the next instruction in sequence to the program address register. Consequently, as the order word decoder is completing the processing of the data reading the next order word in sequence is returned and data processing from that order begins under control of the buffer word order decoder.

9.2 *Transfer Sequencer*

Because of the degree of overlap in the central processor a transfer sequencer is enabled when a transfer is to be executed. The transfer sequencer stops the flow of instructions for at least one cycle, until: the transfer address has been placed in the program address register, transmitted to the program store, and the first program order word of the new sequence of instructions has been returned to the central control. In addition, the transfer sequencer controls gating actions required for indirect transfers, and in such instances one or two additional machine cycles are inserted, depending on whether the transfer address is to be obtained from a call store or program store location.

A class of early transfer orders is included in the order structure; the term "early" alludes to the enabling of the transfer sequencer at a time in the machine cycle earlier than that performed for regular transfer orders. The early enablement of the transfer sequencer serves to inhibit the decoders, so as not to carry out the reading or writing operation when the decision is to execute the transfer of program control.

9.3 *Program Store-Correct Reread Sequencer*

When program or data words are read from the program store, checking circuits in central control examine the 44-bit word received and the previously transmitted program address which has been retained in central control for such checking purposes. Output signals from the checking circuit indicate either:

(1) that all checks are passed and central control continues the processing of that word,

(2) a single error is detected in the 44-bit word received, or

(3) an error is detected in the address transmitted, or an even multiple error is detected, or the program store response did not include an all-seems-well signal. The program store correct-reread sequencer responds to conditions (2) and (3) to correct or to reread and recheck the program store response. The program store correct-reread sequencer increments one of two binary counters for each word corrected or reread. The counters are periodically interrogated to determine the rate at which central control is receiving program store responses containing single or multiple errors.

The failure of the program store correct-reread sequencer to succeed on a retry is designated a "reread failure condition," which requires maintenance action, since the repeated trouble condition is not assumed to be due to a transient error condition. The program store correct-reread sequencer carries out several hardware steps leading to these maintenance actions. First, a maintenance interrupt source signal is generated to switch control to the E-level maintenance interrupt program sequences. Second, the program store correct-reread sequencer stores in a special register in central control the information indicating the trouble condition associated with the rereading failure. The contents of this program store error summary register indicate whether the detected trouble condition was due to a double-error condition, an error in the transmitted address, or a failure of the program store to return its hardware check signal.

Each of the sequencers described here consists of an individual counter and gating circuits within central control; however, the actions taken by one sequencer may serve as part of the data processing steps of other sequencers in central control. For example, the program store correct-reread sequencer is activated to correct or reread program order words, but it is also responsive to data readings obtained from the program store by the data reading sequencer and to indirect transfer addresses obtained from the program store by the transfer sequencer. Whenever the data reading sequencer obtains a data word from the program store, the previously described checks are made, and when correction or rereading is required the program store correct-reread sequencer is activated. In such instances the program store correct-reread sequencer inhibits the gating actions of the data reading sequencer and prevents the counter of the data reading sequencer from advancing to the next internal state. The program store correct-reread sequencer then proceeds to carry out the required correction or rereading, and upon conclusion of its remedial actions returns to the inactive state. This last action automatically releases the data reading sequencer, which continues its handling of the

corrected or reread data word. Similar interactions occur between the program store correct-reread sequencer and the transfer sequencer.

9.4 *Accumulator Sequencer*

This sequencer provides gating actions necessary to the completion of a special class of memory reading orders. The orders are those which perform a memory reading and transmit this memory reading to the accumulator system to become one of the operands in an arithmetic or logical combining operation to be completed in the accumulator. Since the logic combining operation requires additional data processing time, the last gating action (the gating of the resulting combination into the accumulator register) cannot be completed in the time framework for the single-cycle instruction shown in Fig. 31. Accordingly, the accumulator sequencer is enabled whenever an order of this class is executed, and it completes its gating action in one additional clock phase. This sequencer differs from the previously described sequencers in that it shares control of gating actions with the decoders and extends the degree of overlap, rather than inserting additional machine cycles. Such additional time is required to process a memory reading when it is transmitted to the accumulator system, and since extended overlap rather than inserted machine cycles is used in this instance, a memory reading order which uses the accumulator as the destination register may not be followed by an order which uses the accumulator as the index register.

9.5 *Peripheral Sequencer*

The peripheral sequencer is similar to the accumulator sequencer just described in that it carries forward the data processing and instructions on an extended overlap basis rather than within inserted machine cycles. It differs from the accumulator sequencer in that its actions extend over a period of nearly two machine cycles to carry out the transmission of commands and addresses to the central pulse distributor and (as required) to all of the peripheral units such as scanners, signal distributors, network controllers, and so on. The sequencer continues to remain active in order to gate check signals and scanner responses from the addressed units, and to generate trouble signals if any of the appropriate checks fail.

The peripheral orders are used in repetitive scanning operations of lines, trunk circuits, and junctors. To achieve efficient use of real time, the maximum scanning rate of one scan for every two machine cy-

cles is provided. In such instances, the peripheral sequencer remains active during the execution of such a sequence of orders and does not return to the inactive state until (1) the scanning sequence has been completed, (2) an error in the response of the central pulse distributor or peripheral unit is detected, or (3) an interrupt intervenes in the execution of the scanning sequence.

9.6 *Sequencer Design*

A total of ten sequencers is contained in central control. The remainder of the sequencers serve in multicycle operations — for retrial of call store reading and writing operations, for special circuit actions and subsequent return from interrupted programs, for stopping and restarting data processing in the standby central control, and for obtaining special maintenance program sequences from the call store.

The ten sequencers perform different data processing steps as required in the execution of program sequences by central control. The sequencers are designed as a collection of individual counter circuits rather than as one large counter. As noted, one sequencer may "call" a second as required to carry out a class of data processing steps associated with the second sequencer. Thus the decomposition of a sequencer counter results in relatively efficient use of the number of internal states required of the counters. Perhaps more importantly, the flexibility of design gained by this decomposition has proved helpful in developing circuit specifications in the face of simultaneously evolving requirements for each of these circuits.

The heart of each sequencer is a synchronous counter. This synchronous counter responds to input signals from the decoders and check circuits and selected phases of the microsecond clock. It is enabled and advanced through various active states at definite times within a sequence of machine cycles. Although the counters are synchronous in the sense just described, the counters are designed as asynchronous counters and do not require delay lines in their feedback loops. This simplification is achieved by requiring that at least two changes of state must occur in any 5.5-microsecond interval of time (unless of course the sequencer is inactive or being inhibited by another sequencer).

An example of a sequencer counter is shown in Fig. 40 along with the time diagram showing the advancing of the sequencer through a succession of its active states.

Assuming the inactive state corresponds to both flip-flops being reset and assuming that an enable signal appears by T0 of a given machine

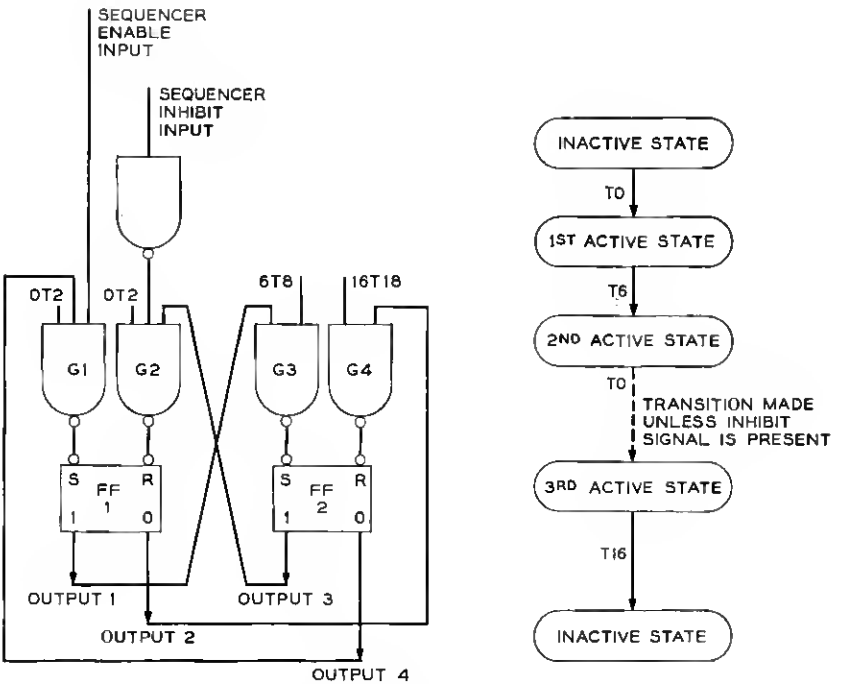


Fig. 40 — Sequencer counter.

cycle, this sequencer is enabled by activating gate G_1 at T0 to set flip-flop 1. This in turn enables output 1 so that gate G_3 is enabled at T6, and the sequencer advances from its first active state to the second active state at this time. Accordingly, by T0 of the following cycle a signal appearing on output 4 will disappear so that gate G_1 cannot be again activated in the immediately succeeding machine cycle. Instead, gate G_2 will be enabled to reset flip-flop 1 unless a signal appears on the inhibit input at this time. If an inhibit signal appears (e.g., the output of another sequencer), the sequencer will remain in the second active state until the inhibit signal disappears and a new machine cycle begins with the reappearance of the clock pulse 0T2.

The example in Fig. 40 illustrates a relatively simple counter design. Some of the more complicated counters, such as the transfer sequencer counter, include alternate sets of active states according to the type of transfer order being processed; the different sets of states serve to carry out different gating actions over different numbers of machine cycles.

The selection of clock pulses for the various counters is made to span

selected phases of various machine cycles to provide the simplest translation of counter state outputs into gating control signals. Where possible, these transitions were placed at such a time as to make sequential lockout of one sequencer to another possible; the extra connection of inhibit signals required in simultaneous lockout schemes is thereby avoided.

Although sequencers seize and release control of each of the decoders at different times (according to the function being performed), the grouping of decoder leads into four classes according to decoders and clock phase suffices for all but a few of the decoder-controlled gates. Accordingly, the outputs of the sequencers are combined to generate four sets of inhibit signals to selectively seize and release control of central control gates for all situations in which the sequencers insert additional machine cycles. The gates that could not be so grouped consist primarily of those which control the flow of program order words from the program store to central control. These had to be handled separately to provide sequencer control gating actions that performed the necessary functions with a minimum number of inserted machine cycles.

X. SUMMARY

This article has described some aspects of the logic design of the central processor, including (1) design considerations, (2) a development of the program order structure, (3) a development of the logic blocks and their interconnections to implement the order structure, (4) a description of order encoding, (5) the circuits and program orders needed to meet maintenance objectives, (6) a discussion of timing requirements, and (7) a description of sequencing circuits required for multicycle data processing functions.

Table IV summarizes the number of components in each of the functional units in one central control unit.[17] One such unit occupies four standard No. 1 ESS bays[6] and requires approximately 2800 printed wiring boards of various logic packages.

This design of the central control represents a balanced compromise between the data processing capability, economy, and reliability of operation in a telephone switching system. Furthermore, the system is able to grow through the addition of modular memory and input-output equipment without extensive wiring changes. The use of overlap operation and the inclusion of special orders to carry out several steps of highly repetitive input-output functions simultaneously assist in obtaining high data processing capability.

TABLE IV — NUMBER OF TRANSISTOR GATES IN FUNCTIONAL UNITS IN ONE CENTRAL CONTROL UNIT

Functional Unit	No. Transistor Gates	Percentage of CC Total
Program store bus circuits	260	2.1
Call store bus circuits	290	2.3
Instruction registers	390	3.1
Decoders (including memory address decoder)	1160	9.3
Error-checking and -correcting circuits and parity generators	640	5.1
Sequencers and sequencer-controlled gates	870	7.0
Index adder system	810	6.5
Program store register, increment circuit, and auxiliary storage register	720	5.8
Mask and complement circuit and insertion mask	280	2.3
Peripheral communication circuits	1140	9.2
Index registers and logic register	1140	9.2
Accumulator system, including shifting and find-rightmost-one circuit	1050	8.4
Masked bus sampling gates (C flip-flops and logic)	80	0.6
Matching circuits	1860	15.0
Emergency-action circuit	170	1.4
Clock circuits	330	2.7
Miscellaneous buffer bus registers, including interrupt sources, central processor status and error summary registers	1010	8.1
Miscellaneous circuits, including power control and maintenance scanning access	240	1.9
Total	12,440 gates	100.0%

XI. ACKNOWLEDGMENTS

Many of our colleagues contributed materially to central control organization. Mrs. E. S. Hoover and I. D. Nehama performed many of the early and fundamental systems studies which led to the present plan, A. H. Doblmaier contributed to the over-all organization of the central control, M. P. Fabisch worked out much of the encoding scheme, R. W. Downing specified most of the maintenance facilities, J. S. Nowak specified the emergency-action circuit, E. Graeve designed the clock circuits, and R. B. Smith and Miss V. R. Smith created the mnemonic representation of orders and the programmer's manual.

REFERENCES

1. Keister, W., Ketchledge, R. W., and Vaughan, H. E., No. 1 ESS: System Organization and Objectives, B.S.T.J., this issue, p. 1831.
2. Biddulph, R., Budlong, A. H., Casterline, R. C., Funk, D. L., and Goeller, L. F., Line, Trunk, Junctor, and Service Circuits for No. 1 ESS, B.S.T.J., this issue, Part 2.
3. Freimanis, L., Guercio, A. M., and May, H. F., No. 1 ESS Scanner, Signal Distributor, and Central Pulse Distributor, B.S.T.J., this issue, Part 2.

4. Danielson, D., Dunlap, K. S., and Hofmann, H. R., No. 1 ESS Switching Network Frames and Circuits, B.S.T.J., this issue, Part 2.
5. Dougherty, H. J., Raag, H., Ridinger, P. G., and Stockert, A. A., No. 1 ESS Master Control Center, B.S.T.J., this issue, Part 2.
6. Cagle, W. B., Menne, R. S., Skinner, R. S., Staehler, R. E., and Underwood, M. D., No. 1 ESS Logic Circuits and Their Application to the Design of the Central Control, B.S.T.J., this issue, p. 2055.
7. Harr, J. A., Hoover, Mrs. E. S., and Smith, R. B., Organization of the No. 1 ESS Stored Program, B.S.T.J., this issue, p. 1923.
8. Carbaugh, D. H., Drew, G. C., Ghiron, H., and Hoover, Mrs. E. S., No. 1 ESS Call Processing, B.S.T.J., this issue, Part 2.
9. Ault, C. F., Gallaher, L. E., Greenwood, T. S., and Koehler, D. C., No. 1 ESS Program Store, B.S.T.J., this issue, p. 2097.
10. Genke, R. M., Harding, P. A., and Staehler, R. E., No. 1 ESS Call Store — A 0.2-Megabit Ferrite Sheet Memory, B.S.T.J., this issue, p. 2147.
11. Ulrich, W., and Vellenzer, Mrs. H. M., Translations in No. 1 ESS, B.S.T.J., this issue, Part 2.
12. Connell, J. B., Hussey, L. W., and Ketchledge, R. W., No. 1 ESS Bus System, B.S.T.J., this issue, p. 2021.
13. Martellotto, N. A., Oehring, H., and Paull, M. C., Process III, A Compiler-Assembler for No. 1 ESS, B.S.T.J., this issue, Part 2.
14. Downing, R. W., Nowak, J. S., and Tuomenoksa, L. S., No. 1 ESS Maintenance Plan, B.S.T.J., this issue, p. 1961.
15. Tuomenoksa, L. S., and Ulrich, W., Coding and Information Identification, IEEE Trans., **67**, 1963, p. 403.
16. Butler, T. T., unpublished work.
17. Menne, R. S., unpublished work.